SAP资深咨询顾问倾力之作

全面解析SAP Web Dynpro for Java开发技术

俞陈霄 赵旭 周世胜 编著

SAP

Web Dynpro For JAVA开发技术详解

SAP Web Dynpro for Java Detailed Development Guide





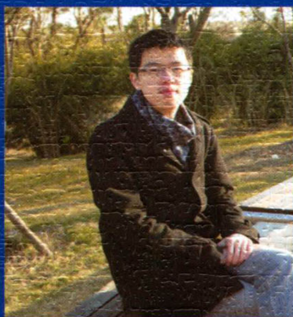


机械工业出版社
CHINA MACHINE PRESS

VISIT…

LANZAROTE
Caliente.com

作者简介



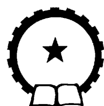
俞陈霄，生于1984年，毕业于同济大学软件学院，SAP资深咨询顾问，拥有美国项目管理协会PMI认证。

自毕业起从事SAP咨询工作，先后供职于康而普、东方航空等公司，担任咨询顾问、运维经理等工作。

对于SAP Web Dynpro for Java语言从技术到架构都有着深入的了解和应用，对于S/4时代的新技术以及原生ABAP语言也有丰富的实战经验。

SAP Web Dynpro for Java
开发技术详解

俞陈霄　赵　旭　周世胜　编著



机 械 工 业 出 版 社

本书详细介绍了 Web Dynpro for Java 的相关开发技术、底层架构及周边拓展应用，以实例贯穿全书，配以丰富的理论说明。前半部分集中对 Web Dynpro for Java 开发技术、底层架构进行详细的描述和讲解，不仅说明了怎么开发，更阐述了为什么这样开发。后半部分主要介绍与 Web Dynpro for Java 相关的各类前沿技术实例以及在新版本中的应用，包括新版本的门户集成、NWA 管理者平台配置、BPM 流程管理建模开发、NW-DI 集成开发环境等内容。

本书可作为 Web Dynpro 开发人员的速查手册，也可作为正在使用相关技术的最终用户以及其他技术顾问学习和了解 Web Dynpro 技术的参考书。

图书在版编目（CIP）数据

SAP Web Dynpro For JAVA 开发技术详解 / 俞陈霄，赵旭，周世胜编著．—北京：机械工业出版社，2017.9
ISBN 978-7-111-58177-2

Ⅰ．①S…　Ⅱ．①俞…　②赵…　③周…　Ⅲ．①企业管理－应用软件－软件开发　②JAVA 语言－程序设计　Ⅳ．①F272.7－39　②TP312.8

中国版本图书馆 CIP 数据核字（2017）第 244006 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
策划编辑：丁　诚　　责任编辑：丁　诚
责任校对：张艳霞　　责任印制：李　昂
三河市宏达印刷有限公司印刷

2017 年 10 月第 1 版・第 1 次印刷
184mm×260mm・26.25 印张・641 千字
0001－3000 册
标准书号：ISBN 978-7-111-58177-2
定价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务
服务咨询热线：（010）88361066
读者购书热线：（010）68326294
（010）88379203

网络服务
机 工 官 网：www.cmpbook.com
机 工 官 博：weibo.com/cmp1952
教育服务网：www.cmpedu.com
金 书 网：www.golden-book.com

封面无防伪标均为盗版

前　　言

在 SAP NetWeaver 出现之前的 30 多年中，SAP 一直在努力使自家的企业管理软件（ERP）产品尽善尽美，从 R/2 到 R/3 再到 ECC，SAP 的核心产品在不断地成熟与蜕变，然而 SAP 的开发技术却始终局限于 ABAP 语言（Advanced Business Application Programming），产品也始终以 C/S 架构存在。SAP NetWeaver 平台的出现是 SAP 为了适应互联网世界的发展做出的一次大胆改革，随之带来的还有全新的 Web Dynpro 技术。

在 SAP 传统 ERP 软件中的屏幕开发技术称为“Dynpro”技术，在决定向互联网敞开怀抱后，新的技术更直接地被定义为了“Web Dynpro”，使用 Web Dynpro 开发的应用程序，都需要通过浏览器访问，简化了安装客户端和熟悉系统的过程。SAP 的 Web Dynpro 同时支持 ABAP 语言和 Java 语言，本书将着重介绍 Web Dynpro for Java（WD4J）技术的相关内容。

从内容上来看，可以将本书分为两部分，前半部分主要介绍 Web Dynpro for Java 的技术细节以及相关开发实例，后半部分则介绍与 WD4J 技术相关的开发内容。Web Dynpro 背靠着强大的 SAP NetWeaver 平台，提供了包括版本管理、质量控制、运行监控以及性能优化在内的一系列服务和功能，而 NetWeaver 平台则依托的是整个 SAP ERP 的大生态圈，这是 Web Dynpro 语言拥有的无与伦比的优势。

本书各章的内容安排如下。

第 1 章：介绍 SAP NetWeaver 架构和 SAP NetWeaver 开发者工作室，一方面了解 Web Dynpro 强大的平台优势，另一方面熟悉开发环境，为学习 Web Dynpro 技术打下基础。

第 2 章：遵循学习开发语言的惯例，先用 Web Dynpro 来完成一个“Hello world”的应用程序，从实战开发开始了解 Web Dynpro 开发过程。

第 3 章：对 WD4J 技术进行分析和讲解，包括视图、窗口、组件控制器、消息池、组件和模型等内容。除了对开发技术的理论介绍外，基本上每一部分内容都会包含对应的开发实例，以方便大家理论与实施相结合。

第 4 章：重点介绍 Web Dynpro 的页面元素。由于 Web Dynpro 的一项优势在于标准化的图形开发并且提供了丰富的元素给顾问选择，因此本章将对常用的一些元素进行介绍，并同样地包含了具体使用案例的讲解。

第 5 章：在熟悉了 WD4J 开发技术以后，本章模拟一部分在实际项目开发中的场景。

第 6 章：介绍 SAP NetWeaver 平台的企业门户，SAP 将其定位成整个平台的统一入口，同时它也是通过基于角色的用户访问 Web Dynpro 应用程序时所使用的系统。

第 7 章：具体介绍 SAP 的业务流程管理，通过 SAP NetWeaver 平台可以实现不同系统间业务的串联，大大提升了用户的工作效率。当然，业务流程的界面开发使用的就是 Web Dynpro for Java 技术。

第 8 章：在介绍了 Web Dynpro 的应用系统企业门户以及最新产品业务流程管理后，不得不提到 Web Dynpro 背后的 NWA 管理者平台，它凭借强大功能为 Web Dynpro 保驾护航。另外，还介绍了其他一些基于 NWA 进行配置管理的开发技术，包括了作业的配置和开发、Adobe 交互表单开发、自定义数据源使用、全局变量应用以及日志管理等内容。

第9章：在大型的开发团队中，SAP NetWeaver开发者架构也是开发团队进行代码管理的一个必要工具，它继承了需要ABAP开发和程序管理中的优秀特点，帮助WD4J的开发人员能够高效地完成各自的开发任务。

本书1~7章主要由俞陈霄编写，赵旭和周世胜参与了编写；第8和9章由周世胜编写；书中涉及的Web Dynpro开发实例均由赵旭设计开发并完成记录，俞陈霄和周世胜参与测试和校验。另外，还要感谢思普艾斯提供的系统环境支持以及其他技术服务支持。

由于编者水平有限，书中难免存在不足之处，望广大读者批评指正！

编　者

目　录

前言
第1章　SAP NetWeaver 架构与开发者工作室 …… 1
1.1　SAP NetWeaver 架构 …… 1
1.1.1　人员集成 …… 2
1.1.2　信息集成 …… 3
1.1.3　流程集成 …… 3
1.1.4　Web Dynpro 技术 …… 4
1.2　SAP NetWeaver 开发者工作室 …… 4
1.2.1　工作室架构 …… 5
1.2.2　功能一体化 …… 6
1.2.3　用户界面和工具 …… 8
1.2.4　Web Dynpro 透视图 …… 9
第2章　Hello World 应用 …… 10
2.1　创建项目 …… 10
2.2　创建组件 …… 13
2.3　配置服务器 …… 15
2.4　部署应用程序 …… 16
第3章　WD4J 工程详解 …… 18
3.1　WD4J 工程简介 …… 18
3.2　视图 …… 20
3.2.1　视图的生命周期 …… 20
3.2.2　视图的描述文件与标准类 …… 23
3.2.3　视图的上下文 …… 26
3.2.4　视图的自定义方法 …… 43
3.2.5　视图的事件处理器 …… 45
3.2.6　视图的 UI 元素 …… 53
3.3　窗口 …… 57
3.3.1　窗口的属性 …… 57
3.3.2　窗口与接口视图 …… 59
3.3.3　视图集与视图嵌入 …… 66
3.3.4　窗口的导航 …… 68
3.3.5　其他窗口类型 …… 70
3.4　组件控制器 …… 74
3.4.1　组件控制器的属性 …… 74
3.4.2　组件控制器的生命周期 …… 75
3.4.3　组件控制器的标准类 …… 77
3.4.4　组件控制器的上下文 …… 78
3.4.5　接口控制器 …… 88
3.4.6　自定义控制器 …… 89
3.5　消息池 …… 90
3.5.1　消息类型 …… 90
3.5.2　消息区域 …… 91
3.5.3　多语言配置（I18N） …… 91
3.6　组件 …… 94
3.6.1　组件的生命周期 …… 94
3.6.2　组件之间的访问 …… 95
3.7　模型 …… 105
3.7.1　模型与自定义控制器 …… 106
3.7.2　RFC 模型 …… 107
3.7.3　Web 服务模型实例 …… 122
3.7.4　EJB 模型实例 …… 144
3.7.5　应用场景贴士 …… 146
3.8　应用程序 …… 147
第4章　用户界面元素 …… 149
4.1　界面元素介绍 …… 149
4.1.1　元素的添加 …… 149
4.1.2　元素的通用属性 …… 151
4.1.3　元素的布局 …… 151
4.1.4　元素设计规则 …… 152
4.2　简单元素 …… 152
4.2.1　按钮 …… 152
4.2.2　文本框 …… 157
4.2.3　关键字下拉列表 …… 161
4.2.4　复选框 …… 165
4.2.5　单选按钮 …… 167
4.2.6　工具栏 …… 169
4.2.7　超链接 …… 171
4.2.8　选项卡集 …… 172

4.2.9 托盘 …… 175
4.2.10 消息区域 …… 177
4.2.11 表单 …… 178
4.3 复杂元素 …… 181
4.3.1 复选框组 …… 181
4.3.2 单选按钮组 …… 183
4.3.3 表格 …… 185
4.3.4 树 …… 205
4.3.5 表格树 …… 211
4.3.6 文件上传 …… 214
4.3.7 文件下载 …… 218
4.3.8 视图容器 …… 222
4.4 特殊元素 …… 225
4.4.1 定时器 …… 225
4.4.2 图片 …… 226
4.4.3 网页框架 …… 228
4.4.4 Web 部件 …… 229
4.4.5 路线图 …… 230
4.4.6 阶段指示器 …… 234
4.4.7 进度指示器 …… 235
4.4.8 菜单路径 …… 238
第 5 章 综合场景应用 …… 240
5.1 前期准备 …… 240
5.1.1 用户需求分析 …… 240
5.1.2 接口服务设计 …… 240
5.2 功能开发 …… 250
5.2.1 创建 WD4J 项目 …… 250
5.2.2 核心功能开发 …… 252
5.2.3 视图元素调整 …… 265
5.3 延伸配置 …… 267
第 6 章 SAP 企业门户 …… 269
6.1 门户与企业门户 …… 269
6.1.1 门户 …… 269
6.1.2 SAP 企业门户 …… 269
6.2 内容管理 …… 271
6.2.1 门户内容介绍 …… 271
6.2.2 门户页面创建 …… 272
6.2.3 门户角色的创建和分配 …… 274
6.2.4 多级菜单配置 …… 276
6.2.5 多样化的视图模板 …… 281
6.3 用户管理 …… 283
6.3.1 账号管理 …… 283
6.3.2 批量管理 …… 287
6.3.3 用户管理 API 开发 …… 289
6.4 系统管理 …… 296
6.4.1 UME 配置 …… 296
6.4.2 内容传输 …… 299
6.4.3 系统监控 …… 302
第 7 章 SAP 业务流程管理 …… 303
7.1 BPM 系统架构 …… 303
7.2 BPM 流程设计 …… 304
7.2.1 模型设计 …… 304
7.2.2 任务定义 …… 307
7.2.3 启动方式选择 …… 310
7.3 BPM 流程桌面 …… 314
7.3.1 通用工作清单 …… 314
7.3.2 任务执行 …… 320
7.3.3 流程监控 …… 321
7.4 BPM 开发实例 …… 322
7.4.1 创建 BPM 流程 …… 322
7.4.2 关联 WD4J 项目 …… 328
7.4.3 映射流程上下文 …… 334
7.4.4 配置启动方式 …… 337
7.4.5 运行测试流程 …… 338
7.4.6 流程监控 …… 339
第 8 章 NWA 相关技术 …… 342
8.1 NWA 管理者平台 …… 342
8.2 服务器实例与应用 …… 343
8.2.1 Java 服务器实例 …… 344
8.2.2 Java 服务和应用程序 …… 345
8.3 作业与定时任务 …… 346
8.3.1 创建作业项目 …… 346
8.3.2 NWA 配置定时任务 …… 348
8.3.3 WD4J 创建定时任务 …… 350
8.4 Adobe 技术交互表单 …… 351
8.4.1 配置本地 LCD …… 351

8.4.2 配置 ADS 服务 …………………… 353
8.4.3 WD4J 开发交互表单 ………… 355
8.5 自定义数据源 …………………… 361
8.5.1 创建连接驱动 ………………… 361
8.5.2 配置自定义数据源 …………… 363
8.5.3 WD4J 连接实例 ……………… 364
8.6 Java 属性类的应用 …………… 370
8.6.1 设计时配置 …………………… 370
8.6.2 运行时调整 …………………… 373
8.7 日志管理 ………………………… 374
8.7.1 设计时日志记录 ……………… 374
8.7.2 运行时日志查看 ……………… 375
8.7.3 日志级别配置 ………………… 376
第 9 章 开发基础架构 …………………… 379
9.1 NWDI 的组成 …………………… 380
9.1.1 设计时资源库 ………………… 381
9.1.2 组件编译服务 ………………… 382
9.1.3 变更管理服务 ………………… 385
9.2 NWDI 初始化配置 ……………… 388
9.2.1 CMS 配置向导 ……………… 388
9.2.2 SLD 命名保留 ……………… 390
9.3 NWDI 开发应用实例 …………… 391
9.3.1 导入开发配置 ………………… 392
9.3.2 创建开发组件 ………………… 393
9.3.3 检入与激活 …………………… 394
9.3.4 协同开发场景 ………………… 397
9.3.5 代码冲突 ……………………… 403
9.4 开发组件的依赖 ………………… 406
9.4.1 公共部分与依赖明细 ………… 406
9.4.2 依赖规则参考 ………………… 407

第 1 章　SAP NetWeaver 架构与开发者工作室

Web Dynpro for Java（简称 WD4J）开发技术的优势在于拥有 SAP NetWeaver 平台以及 NetWeaver 开发者工作室，前者是 WD4J 的坚实后盾，而后者则是 WD4J 披荆斩棘的利剑。本章将对它们进行介绍，为之后的 Web Dynpro 开发做好准备。

1.1　SAP NetWeaver 架构

SAP 公司专注于 ERP（Enterprise Resources Planning，企业资源计划）软件数十年，随着不断进行需求延伸以及功能拓展，它在传统的 ERP 软件产品外还研发了客户关系管理系统（Customer Relationship Management，CRM）、供应链管理系统（Supply Chain Management，SCM）、供应商关系管理系统（Supplier Relationship Management，SRM）和产品生命周期管理系统（Product Lifecycle Management，PLM）等产品，以及与之相关的诸多行业的成熟解决方案。然而，SAP 不仅向客户提供了成熟的业务应用程序或系统，在每一代 SAP 成熟套装软件中还提供了与之相匹配的工具、框架及服务以支持在此基础上的开发和配置工作，SAP NetWeaver 正是这一代产品的基础。

除了为相关 SAP 产品提供框架服务和支持以外，SAP NetWeaver 本身还是一个集成化的平台，提供了传统 ERP 无法完成的集成和整合功能。通过 SAP NetWeaver 的架构图，就能看到 SAP 为了从各个层面对业务、数据甚至流程进行集成而做的努力，如图 1-1 所示。

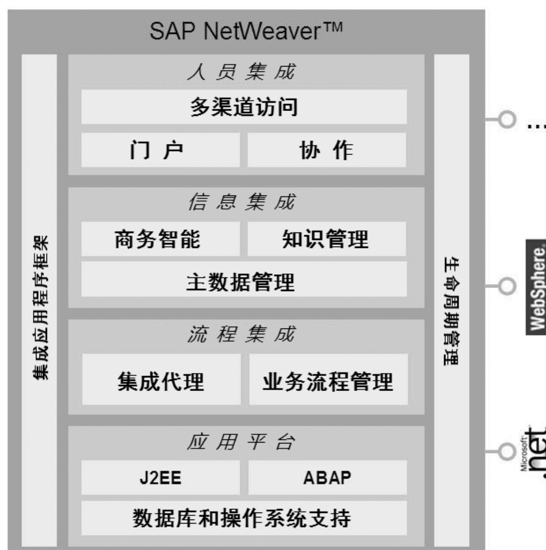


图 1-1　SAP NetWeaver 架构图

SAP NetWeaver 应用平台同时支持 JavaEE 架构和 ABAP 架构，针对套装软件中的不同产品也需要选择不同类型的应用服务器（Application Server，AS）安装，这样既保留了 SAP 传

统的技术特点，又逐步地向 JavaEE 的方向靠拢。图 1-1 中上方的 3 个集成层分别代表了“人员集成”“信息集成”和“流程集成”，所有集成应用都建立在底层的应用平台之上。需要注意的是，这样的 SAP NetWeaver 组件架构并非表示几个集成层面有着继承关系，事实上每一层的集成应用都是相对独立的，而所有的组件可以组合在一起提供完善的功能和服务。

1.1.1 人员集成

在 SAP NetWeaver 的架构中，人员集成包含了多渠道访问、门户和协作等内容，这些内容都有一个相同的前提条件，就是所有系统用户都拥有一套标准化的基于角色设计的访问方式，并且用户可以通过不同的方式进行访问，如移动端或是个人计算机端。而企业门户则将所有的业务系统入口进行集中，并且通过相关的配置实现用户的单点登录（Single Sign - on，SSO），即用户只需要进行一次登录动作就可以访问所有的相关系统。在传统的企业信息化系统大环境中，用户需要登录若干不同的系统进行相关业务的处理。例如，一位部门经理需要登录到人力资源管理系统来访问所有的人事相关信息，包括人员报表、薪酬统计、绩效考核等内容，接下来又需要登录对应的财务系统对部门的财务状况进行汇总分析，可能还需要登录另一个仓储管理系统来查看和审核相关的仓储调配。SAP 企业门户的任务就是将这些入口统一起来，这样用户不再需要了解每个系统的具体情况、登录方式或界面操作，只需要登录企业门户，就可以通过菜单和链接在系统内进行查看和处理所有相关系统的各项业务。

与用户在企业中的位置一样，门户集成也同样基于角色的概念，因为每个用户都在企业中有相应的岗位和角色，如系统管理员、部门经理、采购员或销售员。门户中的每个角色都包含了对应岗位的所有的功能以及信息，且可以灵活地进行组合和拆分，方便应对不同企业的业务状况。面对不同类型的系统，门户同样提供了丰富的框架和服务，用于支持不同类型的页面，这其中不仅包含简单的 HTML 网页，还包括了 Web Dynpro 开发的应用程序、数据仓库报表、水晶报表、SAP GUI 界面以及 Adobe 交互式表单等类型。

在传统 ERP 软件中，产品提供了不同的功能，用户在操作对应业务时需要首先访问不同的功能，然后才能在功能页面中操作与自己业务相关的内容（如 ECC 系统中的事务代码）。相对地，门户在整合了复杂的页面后，一方面提供了页面和多样化的布局对页面进行整合，以便对这些页面进行组合或者分类；另一方面，门户还提供了工作集和角色用于构建复杂的目录层次，最终每个用户都会根据分配的角色内容看到各自对应的导航目录以及页面内容。个性化定制的页面可以最大限度地提升用户的参与度，并降低学习和培训难度。

在人员集成层的协作其实是与信息集成层的知识管理紧密相连的，且这两个组件都通过 SAP 企业门户提供相应的功能和服务。这是因为在企业中，不仅拥有海量的结构化数据，还有大量的非结构化数据，如各种各样的文档、图片或者邮件存档等内容，这些需要通过系统进行管理和访问。通过数据仓库的管理，结构化的数据已经可以通过门户的应用程序进行读取和展现或者通过数据仓库的报表进行浏览，所以非结构化数据在门户中的访问和使用也就变得十分必要了。知识管理并非仅仅是一个文档管理的工具，它同时还具有内容管理的功能，配合门户中的搜索引擎，可以使用户方便、迅速地定位到想要查找的文档和内容。

因为文档的主要功能是用于交换信息，所以用户经常需要访问一些共享文档，为了这样的业务需求，SAP 同样在门户中提供了许多与协作相关的功能和服务。所有用户的交流发生

在门户的协作工作室中，类似网络上的论坛。从技术角度来看，每个工作室其实是一个文档容器，同时也可以集成很多与信息项目管理相关的内容，如日志、任务清单等内容。

所有这些功能和服务都努力地将企业用户在使用 ERP 软件以及 ERP 相关的周边产品时集中到了一起，也为企业以及企业中的每个用户进行业务操作、报表查询以及流程处理带来了许多便利。

1.1.2 信息集成

在 SAP 的产品之中，除了 ABAP 架构的 ERP 产品之外，数据仓库（后来逐渐进化成了商务智能系统）是 SAP NetWeaver 中最成熟稳定的组件。数据仓库的作用是什么呢？打个比方，企业的主数据形成了企业的骨架，而其他的业务数据则构成了企业的肌肉，通过业务数据分析可以了解到潜在客户和潜在的市场，可以优化企业的物资分配和库存，还可以帮助企业更有选择性地进行采购并控制成本。数据仓库的作用就是用于管理除主数据、非结构化数据之外的所有商务内容。

在最初的阶段，企业的业务数据往往分布在若干不同的业务系统中，所以 SAP 开始构建一个数据仓库，它的数据并不是由业务员手工录入的，而是从其他相关的业务系统中抽取并整合在一起的。为了使这个数据仓库对于所有业务数据的分析和评估尽可能地灵活和全面，并且尽可能地优化数据读取和展现的性能，最终 SAP 决定从 OLTP（Online Transaction Processing，联机事务处理）的系统中抽取数据，并经过 SAP 数据仓库的处理存放在适用于 OLAP（Online Analytical Processing，联机分析处理）应用程序的数据结构中。在后来的发展过程中，为了优化报表展示的效果，SAP 收购了 Business Object 产品，并将其与数据仓库进行了深度的融合。

在信息集成层，除了数据仓库之外，主数据管理是 SAP 后来添加到 SAP NetWeaver 的一个组件。它主要用于管理企业系统间的共享数据，包括客户、供应商、账户和组织部门相关数据等内容。由于强大的 SAP NetWeaver 平台支持，主数据管理系统也支持不同的技术框架和接口类型，用于和其他业务系统进行数据集成，同时它也支持 Web Dynpro 技术的应用程序或接口的开发。

1.1.3 流程集成

流程集成主要针对的是跨业务系统流程的串联和整合，而并不仅仅针对需要审批和推送待办任务的流程管理，所以在 SAP NetWeaver 平台中流程集成对应了两个重要的产品和内容：SAP 交换架构（Exchange Infrastructure，XI）以及业务流程管理系统。其中 SAP 交换架构后来升级并直接改名为流程集成系统（Process Integration，PI）。

流程集成系统的作用主要是用于同步和异步消息的交换，并且帮助 SAP 相关系统与非 SAP 系统的数据交互，最终将原本复杂的网状业务系统互联场景替换为更为简洁的星形架构，即所有的系统数据、消息都面向 PI 进行发送，并由 PI 系统统一进行转发。PI 系统针对不同类型的接口提供了多样化的适配器进行连接，同时也规范了所有外围业务系统的接口定义和数据传输。由于 SAP NetWeaver 平台的支持，PI 系统也有完整的数据传输记录和日志查询功能，提高了业务数据传输的安全性和可靠性。

PI 系统本身只能处理简单的流程，在实际业务场景中，更多的是对周边多样化的业务系统数据和接口进行统一，或是将多个系统的业务数据集中到一个系统中，方便业务人员进

行处理。而相对地，业务流程管理系统（Business Process Management，BPM）则是将不同系统的数据集中进行处理和审核，并实时推送待办任务的一个流程管理系统。如果说门户系统方便了用户查看报表，处理单一任务，那 BPM 的流程管理则帮助用户将不同业务系统的处理逻辑集中到了一起，在审批过程中支持会签、委派等功能，在流程流转过程中还具备了完善的监控系统，提供了完整的系统运行日志，保证业务流程顺利运行。

1.1.4 Web Dynpro 技术

在 SAP NetWeaver 的众多组件中，基于 Java AS 运行的应用程序、服务、功能和系统大多是通过 Web Dynpro for Java 技术实现的前端页面。在传统的 ERP 软件中，所有的界面都是以功能为导向进行开发。例如，界面 A 可以完成功能 1 和 2，而界面 B 可以完成功能 3，用户为了完成不同的功能，需要主动地访问对应的功能页面（也就是 SAP ECC 中事务代码的概念），同时 ABAP 语言的开发也需要学习和培训才能了解用户界面的组成和使用。然而在以 Java 应用服务器为基础的系统中，所有界面元素开发都通过 Web Dynpro 的图形化界面进行展现，即使是不了解任何开发语言的业务人员，也能对界面提出相应的要求，达到所见即所得的效果。另外，Web Dynpro 开发的应用程序通过页面集成方式添加至门户中，并以面向角色的方式添加给不同的用户或用户组，使得用户在登录系统后就能看到对应角色或者是对应权限所能使用的所有功能，这为传统的 ERP 用户带来了全新的体验。

1.2 SAP NetWeaver 开发者工作室

在介绍 Web Dynpro for Java 的集成开发环境（Integrated Development Environment，IDE）之前，首先来比较一下 ABAP 工作台与传统 Java 开发 IDE 的区别。

ABAP 工作台的特点如下：

- ABAP 工作台的开发是面向服务器端的。
- 所有的设计开发工具都可以通过整个开发框架（事务代码 SE80）来实现。
- 包含编译、发布以及版本管理的完善的基础架构与 ABAP 工作台无缝集成，同时支持大规模协作开发。
- 支持 SAP 相关的技术标准，包括远程功能模块调用、BAPI 接口公开以及 ERP 系统内各种形式开发等技术内容。

传统 Java IDE 的特点如下：

- Java IDE 的开发是在本地进行的，在搭建对应的环境后，开发工具还实现了标签高亮、关键字自动完成以及本地编译等功能。
- 可以搭配本地的运行环境搭建独立的测试环境。
- 支持一些开放的技术标准，如 J2SE、J2EE、HTML、XML 等技术。

在两者开发环境比较之后，SAP 决定在 WD4J 的开发工具中将两种开发手段的优势集中到一起，它将会满足：

- 为所有开发设计工具提供统一的框架，并同时集成所有的开发架构组件。
- 为大规模开发的团队提供便利的集成性架构，提供统一的编译运行环境以及版本管理。

- 提供本地化开发的支持，包括代码关键字自动完成以及本地编译等工作。
- 优化支持开发的技术标准以及 SAP 的相关技术标准。
- 支持多样化的工程创建和编译，支持所有与 SAP NetWeaver 相关开发对象的设计。
- 对于用户界面开发，使用图形化方式进行编辑。

最终 SAP 完成了为 WD4J 开发技术及相关内容开发的 IDE“SAP NetWeaver 开发者工作室”（SAP NetWeaver Developer Studio，SAP NWDS）的框架搭建。其中基础平台和 J2SE 工具集部分都是基于开源的 Eclipse 平台，而为了满足 SAP 对于 ABAP 开发优势的迁移，额外增加了许多组件，包括应用模型生成器、视图设计器、Java 字典、J2EE 工具集，以及新增的抽象层模型和 SAP 服务层的功能支持。开发者工作室架构图如图 1–2 所示。



图 1–2　开发者工作室架构图

1.2.1　工作室架构

下面具体介绍 NWDS 的体系架构。

作为 NWDS 的核心内容，Eclipse 平台是整个 IDE 的基础，它控制着 IDE 的核心功能，同时也提供一个强大的架构，以支持在此平台上的服务扩展和功能延伸。具体来说，一方面，由于 Eclipse 本身与 Java 语言的紧密联系，软件对于本地 Java 开发所拥有的优点都同样地呈现在 WD4J 的代码开发中；另一方面，由于它允许不同软件厂商或开发技术以一定的标准对平台功能进行扩展，在集成 SAP NetWeaver 相关的一些工具集和服务之后，NWDS 已经可以支持几乎所有的 NetWeaver 平台开发类型和内容（如 Java、C、JSP、HTML、XML 等），并且相关的功能（如创建 EJB，作业，Web 服务等）在使用时也与 Eclipse 原本的平台无缝集成。

对于平台的拓展性，新的功能和工具集都在基于 Eclipse 的“Plug – in”技术标准上进行集成，每一个插件都可以作为一个单独的功能进行开发、拓展并集成至 Eclipse 平台中，较为复杂的功能也可以通过多个插件组合实现。从某种程度而言，Eclipse 平台本身即是由若干大大小小的插件组合而成的，如提供核心服务功能的资源管理子系统和工作台。作为 SAP WD4J 的开发工具，NWDS 正是在这些标准功能之上扩展了许多的工具和服务，并且相应地提供不同的视图以及透视图给不同开发任务的开发人员进行开发和使用。

对于 SAP 在 NWDS 中所做的扩展功能，SAP 既努力保证对原有核心功能和服务优势的利用，同时也尽可能地将不同的新拓展功能使用同一标准的操作页面进行集成，减少了开发人员学习和培训的时间成本。例如，所有的开发组件（Development Component，DC）项目都是基于元数据进行创建和开发的，所有的 DTR（Design Time Repository）相关开发动作都拥有同一标准的操作方式和显示方式。另外，还有集成在 NWDS 中的许多向导功能或 Web Dynpro 开发内容自动创建功能，也都是经过 SAP 的标准化和优化后的结果。

通过这一系列的努力，最终完成了 SAP NetWeaver 开发者工作室这个 WD4J 开发平台——一个基于开源的 Eclipse 软件，同时具有稳定、全面集成等特性的开发架构的集成开发环境。

1.2.2 功能一体化

NWDS 除了拥有平台和架构的优势之外，还适用于大型 SAP NetWeaver 项目和团队协作开发，并在 Java 开放式平台的基础上（J2SE、J2EE、XML、Web 服务等）支持 SAP 相关技术（Web Dynpro、Java 字典等）以及 SAP NetWeaver 开发架构。

1. 与 SAP 开发架构集成

NWDS 除了提供丰富的工具集以支持与 SAP 不同类型项目相关的 Java 开发以外，还与强大的 SAP NetWeaver 开发架构（NetWeaver Development Infrastructure，NWDI）进行了深度集成，使其成为了一个能够覆盖整个 WD4J 项目开发周期的复杂高效的开发场景（见图 1–3）。这一功能的优势在需要大规模开发团队同步进行开发工作时尤为明显。

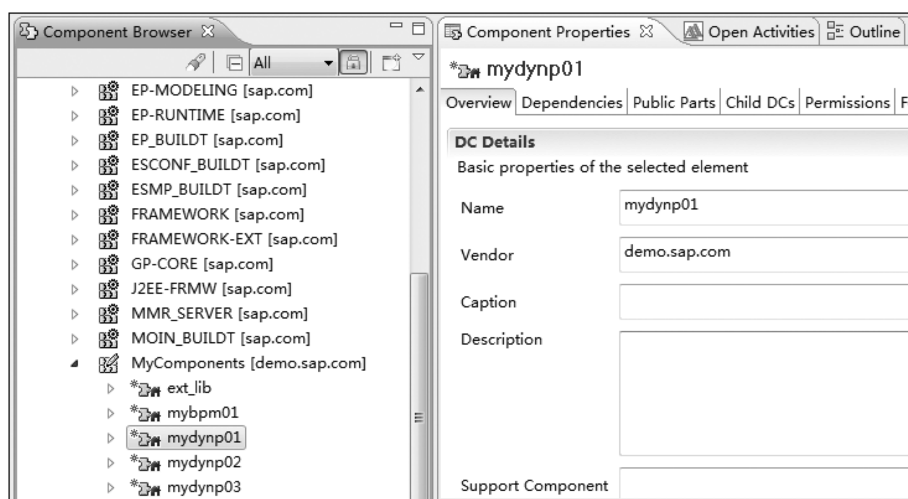


图 1–3 NWDS 中开发基础架构透视图

大规模的开发团队需要将整体开发内容划分成不同的开发模块，并分配至不同的开发小组，然而不同小组间的开发也有可能有代码重叠的部分，这就需要 NWDS 可以满足统一的开发环境、代码同步以及版本管理功能。

当然，作为以 Eclipse 为核心的开发工具，NWDS 也同样对本地的开发过程提供强大的支持和帮助。对于本地的开发过程，并不需要应用 Java 开发架构（Java Development Infrastructure，JDI），但通过 NWDS 的工具版本以及资源库的统一，不同的本地开发人员同样可以在各自的个人计算机上拥有统一的开发环境。在开发完成后，也可再次地与 SAP 开发架

构资源库进行同步，使不同开发方式的需求都能被满足。

以下是在实际 SAP 项目中的一个模拟开发过程：

1）当一名开发人员在 NWDS 中打开一个项目时，首先需要保证本地的项目相关组件、类库以及 Java 源文件可用，这会在打开 NWDS 的过程中自动进行检查。

2）开发人员可以进行必要的修改，并完成本地的一个新的编译版本，也可以通过连接测试服务器对本地的应用程序进行发布和测试，测试的前提是在 NWDS 的选项中正确配置 J2EE 服务器的相关内容。

3）当完成本地的单元测试以后，只要确认可以检入到 DTR 资源库中，就可以通过 NWDS 中的 SAP 开发架构将代码同步至代码资源库。

4）成功上传代码后，开发人员也可以通过 NWDS 将对应的修改进行激活，相应地，开发对象也会从“未激活”状态变为“激活”状态，这个过程需要服务器端中央编译服务的支持。然而与传统的集中编译相比，节省了需要服务器集中编译的处理时间，因为在开发人员发出操作请求的同时，可以实时地对修改过的开发对象进行编译和激活。如果在这个过程中发生错误，开发人员从 NWDS 端就能看到实时的错误消息，方便联系相关人员及时地进行故障排除。

5）在所有的测试以及激活动作成功完成后，修改的相关开发对象就可以被释放并进行传输了。同样地，释放和传输也都可以通过 NWDS 端在相应的视图中操作完成，而这一过程也同样需要服务器端 SAP 开发架构的支持。

2. 与服务器端集成

对于 Eclipse 开发工具而言，更多地用于本地 Java 项目开发，但 NWDS 则不然，对于项目中拥有较大权限的开发人员来说，他们同样也可以通过 NWDS 对 SAP Java 应用服务器直接进行服务器指令的操作和执行。

在执行服务器指令前，首先需要进入 NWDS 的“Management Console”，选择添加 SAP 服务器，并在弹出的对话框中输入正确的服务器连接信息，如图 1-4 所示注意，必须使用操作系统管理员账户进行连接，同时，请勿泄露管理服务器管理员账户信息。

在连接成功后，就能在对应视图中看到服务器的运行实例、运行状态以及硬件使用情况等信息，如图 1-5 所示。

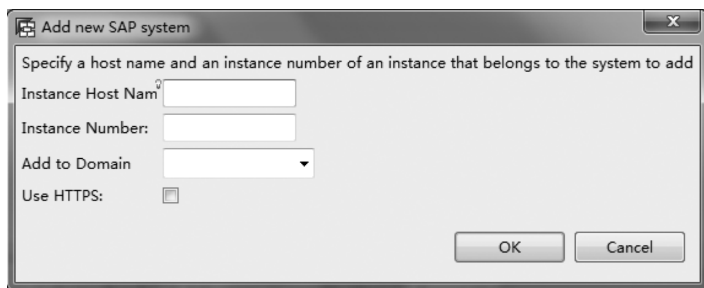


图 1-4　输入服务器连接信息对话框

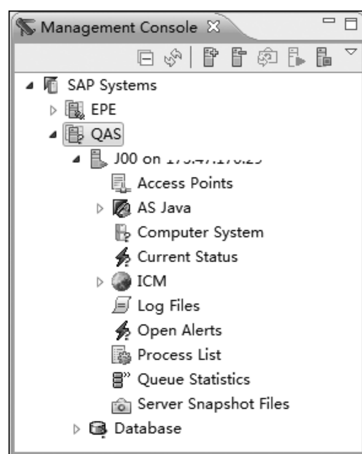


图 1-5　管理控制台视图

然而在这个视图中，不仅仅提供给开发人员查看服务器的相关信息，使用相应权限的账号连接后还可以通过右击服务器实例弹出的快捷菜单对服务器进行启动、停止和重启等操作。而对于开发人员来说，比较常用的是切换服务器实例的 DEBUG 模式，即允许本地应用程序在服务器端的断点调试功能。

除了对服务器实例的操作，还可以通过“Deploy View”和“Undeploy View”使用手动方式对本地应用程序包（EAR 包）进行部署和卸载等操作，而在相应的视图中也可以实时地查看服务器端应用的运行情况是否正常，使核心开发人员只需要专注于 NWDS 的使用，提升了整体的工作效率。

1.2.3 用户界面和工具

NWDS 基本的界面架构与 Eclipse 类似，SAP 也是在尽可能保持界面操作与 Eclipse 一致的前提下添加一些与 NetWeaver 架构相关的功能和服务。概括地说，用户界面由透视图、视图以及编辑器组成，每一个组件都是通过一个子窗口进行展示，而当多个视图或编辑器重叠时，会以标签页的形式出现在一个窗口中，每次仅有一个视图可以进行访问和操作。

透视图是由对应的工具栏以及若干视图和编辑器组成的一个窗口集合，不同透视图的默认布局都是根据该透视图的作用和开发内容进行设计的。当然，使用者也可以在 NWDS 的高级选项中对不同透视图的布局和显示内容进行个性化的调整。当需要使用默认的透视图布局显示时，也可通过单击“Window”→“Reset Perspective”命令，对透视图布局进行重置。对于新接触 NWDS 的使用者来说，透视图默认包含的内容都覆盖了对应开发内容的标准功能和服务，建议在熟悉相关操作后再尝试着进行透视图的个性化设置。

编辑器是在编写代码时打开的窗口，通过编辑器开发人员才可以对开发对象的代码进行修改和更新。在 NWDS 中，默认会对代码的编译错误进行实时地提醒，但并不会对代码进行实时地保存，需要通过工具栏中“保存”按钮或菜单中的“保存”命令对修改的开发对象进行及时保存。除了 Eclipse 标准支持的 Java 代码以及文本编辑之外，NWDS 还提供了一些额外的编辑器。例如，EJB 编辑器用于查看和修改 EJB 的对应配置，并可通过按钮直接跳转至对应接口或 Java 类的代码；Web Dynpro 视图编辑器提供开发人员以图形化的方式进行 Web Dynpro 视图的设计和开发。

视图则是用于查看开发对象相关的各种信息，最常见的就是通过一个树状结构展示项目结构以及项目包含的开发对象，或者显示 Web Dynpro 视图中元素的结构，或者显示开发对象的属性、编译错误消息以及运行日志等内容。开发人员可以通过单击“Window”→“Show View”命令，方便地在当前透视图中添加自己想要查看的视图。在大部分的透视图中，左上窗口显示项目的结构及包含的开发对象，右上窗口用于打开开发对象的编辑器，左下窗口显示开发对象的结构（如 Java 类结构或 Web Dynpro 视图结构），右下窗口则显示属性、日志等相关信息。图 1-6 显示的是 Web Dynpro 的透视图，默认共有 4 个窗口显示在主界面中，每个窗口中又分别叠加了若干的视图在其中。

NWDS 中提供了类型丰富的透视图，对应了不同的开发需求以及工具集的使用，而大部分的透视图的使用与传统的 Java 工程开发类似，所以在这里只重点介绍一下 Web Dynpro 透视图，不再对其他各个透视图一一介绍，这是为读者开展后续的 Web Dynpro 开发工作做好铺垫。

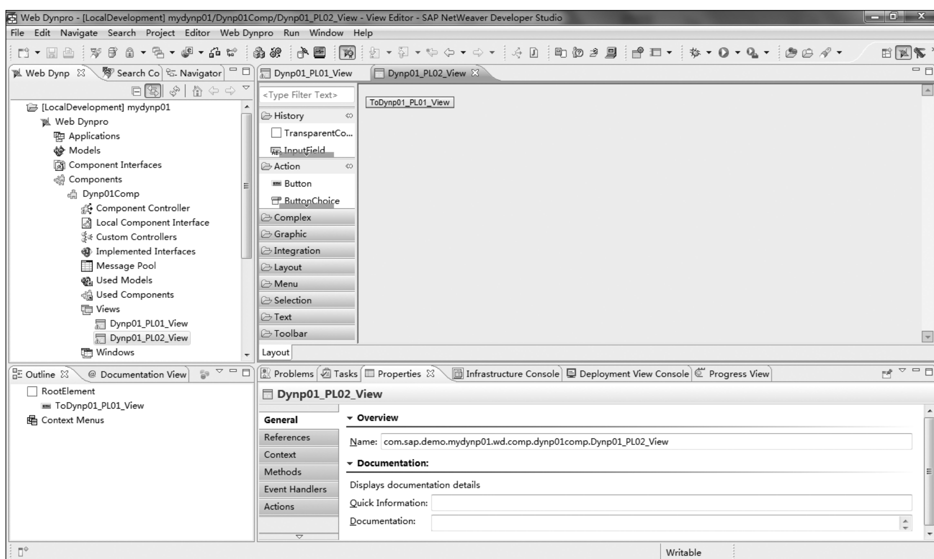


图 1-6　NWDS 中的 Web Dynpro 透视图

1.2.4　Web Dynpro 透视图

Web Dynpro 技术作为基于 SAP Web AS 的新一代编程技术，不仅提供必要的运行服务以及元数据模型的支持，还提供了一系列设计开发工具集成在 NWDS 里的 Web Dynpro 透视图中，为开发人员整个项目周期的工作提供支持和保障。这其中包括了以下内容：

- 在开发环境中提供丰富的功能菜单，如创建和生成项目组件、导入和应用模型，或发布并运行测试应用程序等功能。
- 大部分的 Web Dynpro 开发内容都可以通过向导的形式，如不同类型的 Web Dynpro 模型导入，组件的创建以及对应视图和窗口的生成，通过应用模板方式创建视图元素等。
- 对于应用程序界面的开发使用图形化编辑器，一方面支持视图元素的图形化支持，另一方面只是元素对应逻辑事件的自动生成，尽可能地将开发人员的代码编写工作最小化。

图 1-6 是 Web Dynpro 透视图的实际截图，除了截图中看到的视图以外，还包含了 Web Dynpro 浏览器视图、导航器视图、视图设计器、模型类设计器以及控制器设计器等功能。

Web Dynpro 浏览器（Web Dynpro Explorer）是 Web Dynpro 透视图的核心部分，它为本地或是基于开发组件的 Web Dynpro 项目提供了一个逻辑结构视图，可以通过浏览器了解 Web Dynpro 项目的具体结构，也可以查看组件和窗口的关系、模型的引用以及组件接口的使用等内容。而在同一窗口中，透视图也提供了一个导航器（Navigator），用于查看项目在实际本地计算机中的目录结构以及开发对象所对应的实际文件清单。

与 Web Dynpro 开发相关的视图设计器、控制器设计器等视图，都可以通过双击 Web Dynpro 中对应的组件自动打开相应的编辑器并进行图形化的操作（如引用关系、视图嵌套等）；对于图形化操作比较陌生的开发人员，可以在属性视图中进行手动配置，两者都能达到相同的效果。

旧版本中的上下文编辑器、方法定义、动作定义等都已集成到了属性视图中，以方便开发人员进行查找和定位。

第 2 章　Hello World 应用

在了解了 Web Dynpro 技术的平台支持以及工具支持之后，下面介绍“Hello Word 应用”，并带领大家进入 Web Dynpro for Java 的新世界。

2.1　创建项目

1）首先运行开发工具 SAP NetWeaver 开发者工作室（NWDS）的应用程序“SapNetweaverDeveloperStudio. exe”，在启动过程中 NWDS 会自动加载必要组件并进行开发环境的检查，如图 2-1 所示。由于实例中使用的测试服务器是 NetWeavero 7. 4 版本，因此开发工具需要使用对应的“EHP1 for NWDS 7. 3”版本。



图 2-1　开发者工作室的启动界面

2）默认进入欢迎界面，如果是首次运行，会弹出一个 JVM 的报错消息（见图 2-2），这是由于 NWDS 的运行环境必须先指定对应的 JDK 路径。单击“OK”按钮。

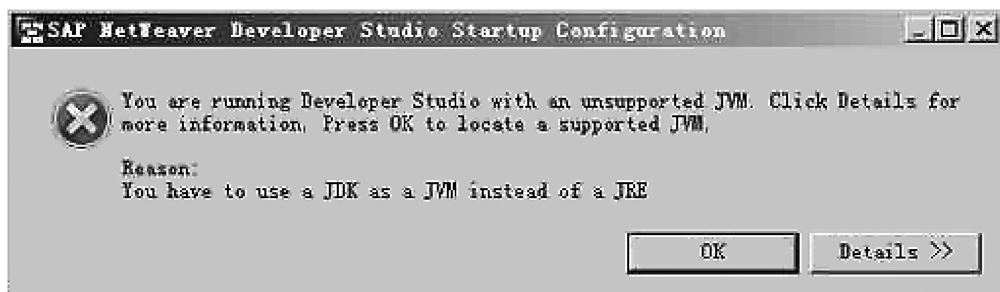


图 2-2　首次启动时报错消息框

3）在弹出的对话框中对应本地计算机中已安装的 jdk 路径，找到“javaw. exe”文件并单击“打开”按钮，如图 2-3 所示。其中 jdk 参考路径为…/Java/jdk1. 6. 0_30/bin。



图 2-3　选择正确的 jdk 路径

4）指定对应的 JDK 后，会弹出另一个消息提醒（见图 2-4），需要重启 NWDS 程序才能使配置生效，单击“Exit”按钮并重新运行 NWDS，以完成 NWDS 初始化配置。



图 2-4　配置完成后提醒消息框

5）再次运行开发者工作室程序后，首先看到的是一个欢迎界面，提供了一些简单的向导以及相关描述信息（见图 2-5），单击“Workbench”按钮返回到工作台主界面。

6）单击“File”→“New”→“Project”命令，创建一个新的项目，如图 2-6 所示。

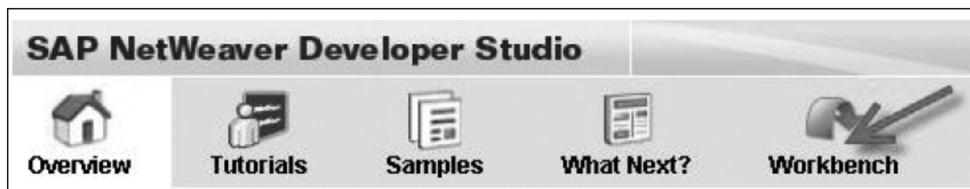


图 2-5　进入 NWDS 后的默认按钮

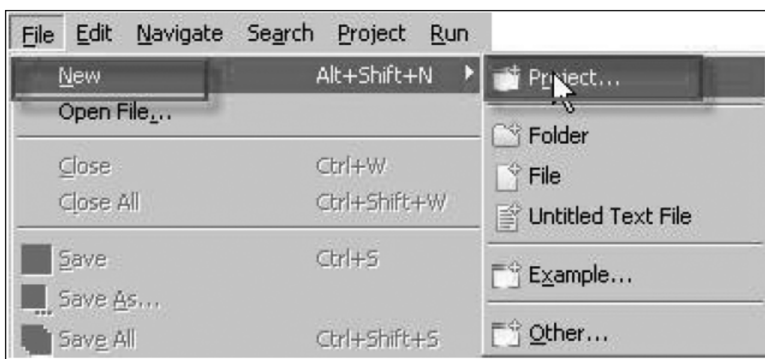


图 2-6　创建一个新项目

7）在弹出的对话框中选择“Web Dynpro Development Component”类型（见图 2-7），单击“Next”按钮。

8）选择在本地开发并在“MyComponents”中进行创建（见图 2-8），单击“Next”按钮。



图 2-7　选择创建项目类型

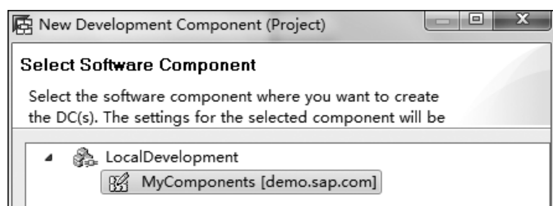


图 2-8　选择开发组件

9）在弹出的对话框中输入项目的名称和标题（见图 2-9），单击“Finish”按钮。

10）至此成功创建了一个新的 Web Dynpro 项目，如图 2-10 所示。

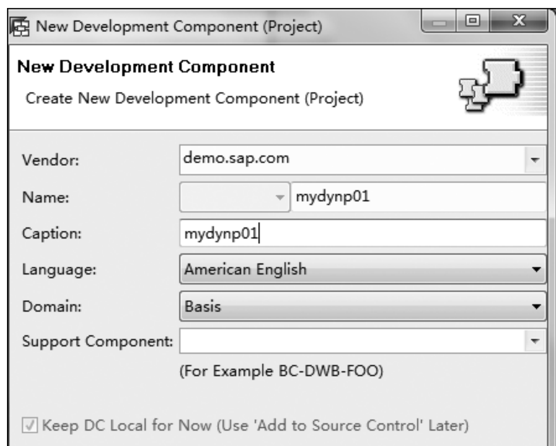


图 2-9　输入项目的基本信息

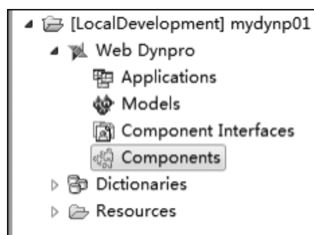


图 2-10　创建完成后的 Web Dynpro 项目

2.2 创建组件

1）如同 Java 工程由包和类组成，Web Dynpro 工程则是由组件组成的。展开项目的清单，右击“ Components”，在弹出的快捷菜单中选择“Create Component”创建一个组件，如图 2-11 所示。

2）输入组件的名称“Dynp01Comp”，选中创建默认的窗口和视图（见图 2-12），单击“Next”按钮。注：在 Web Dynpro 中的很多开发可以通过标准的向导功能完成，在一定程度上减少了手动配置的工作量。

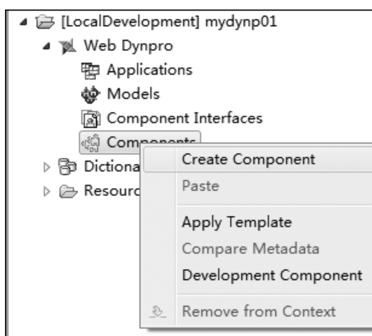


图 2-11 创建一个组件

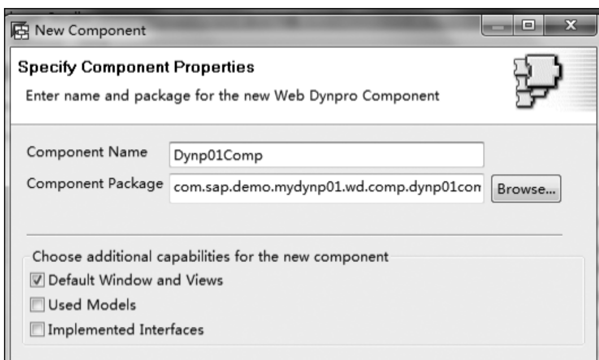


图 2-12 输入组件的基本信息

3）输入窗口的名称“Dynp01CompWindow”以及接口视图的名称“Dynp01 CompInterface View”（见图 2-13），单击“Finish”按钮。

4）在完成组件、窗口和视图的创建后，在 Web Dynpro 浏览器中可以看到一个典型的 Web Dynpro 项目结构，如图 2-14 所示。

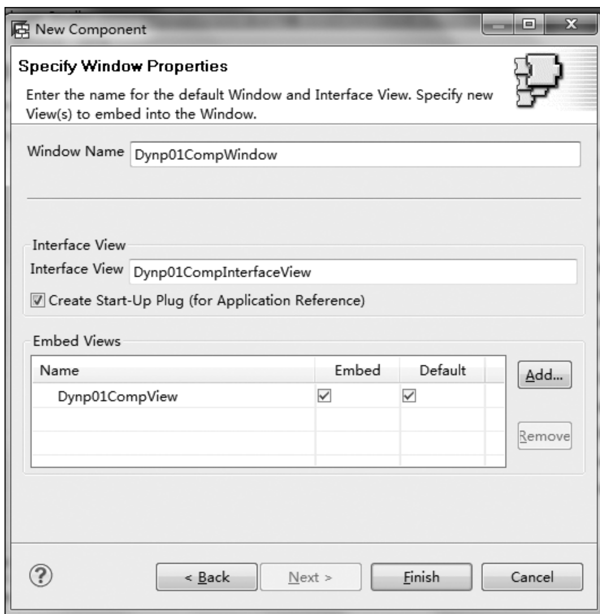


图 2-13 输入窗口和视图的名称

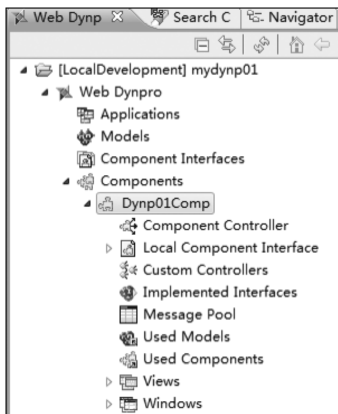


图 2-14 默认组件的目录结构

5）在右侧的模型编辑器中可以看到各个对象之间的关系，如图 2-15 所示。

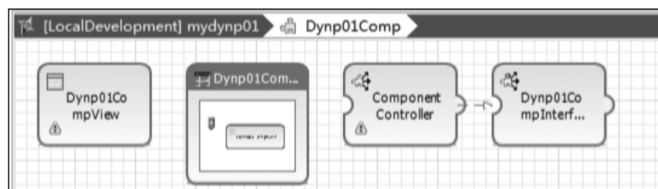


图 2-15　模型编辑器中的对象

6）展开“Views”视图，并双击通过向导创建的视图。在左下角的“Outline”中右击“RootElement”，在弹出的快捷菜单中选择“Insert Child”，插入一个子元素，如图 2-16 所示。

注：每个 Web Dynpro 的视图中都会固定有一个根元素，且无法被删除。

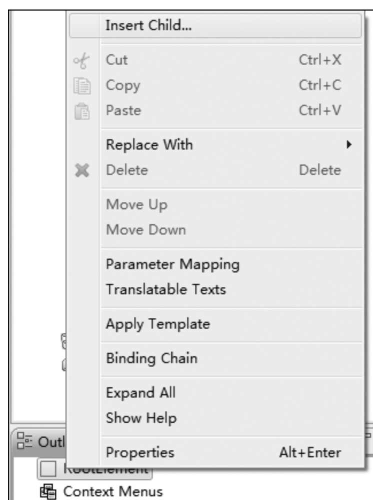


图 2-16　插入子元素的菜单选择

7）在弹出的对话框中选择插入元素的类型，前两个选项区用于显示不同元素类型的目录分类，第三个选项区是具体的元素对象。在“UI Element”选项区中选中“TextView”，单击“OK”按钮，如图 2-17 所示。

8）在右下方的属性选项卡中，将“TextView”的“text”文本内容修改为“This is Hello World Application”，如图 2-18 所示。

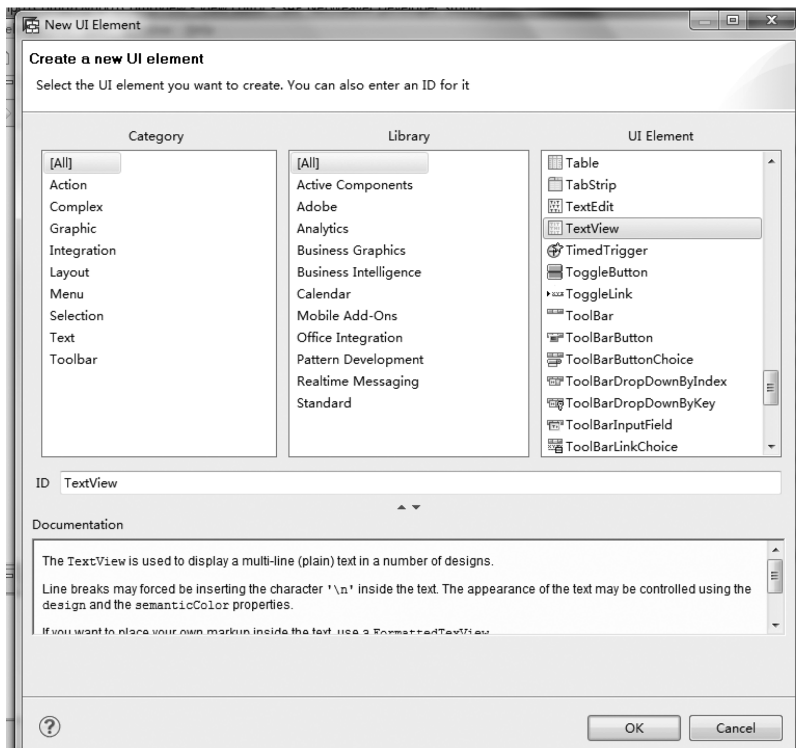


图 2-17　选择元素类型

9）依次将“design”属性修改为“header1”、将“semanticColor”属性修改为“marked2”、将“width”属性修改为“500px”，如图 2-19 所示。完成所有修改后，单击开发工具左上方的“保存”按钮，或按〈Ctrl + S〉组合键进行保存。

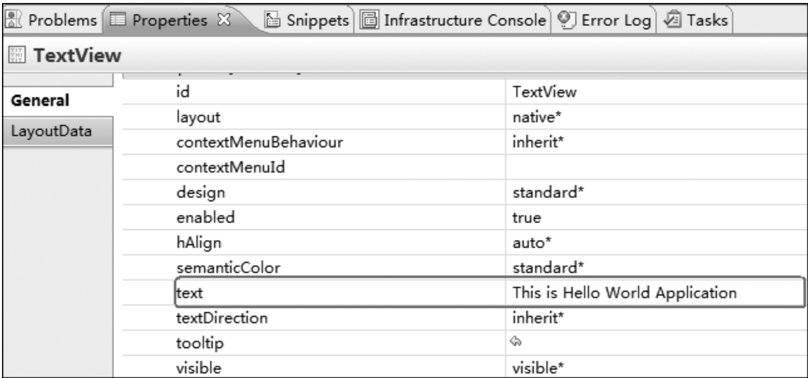


图 2-18　修改元文本属性

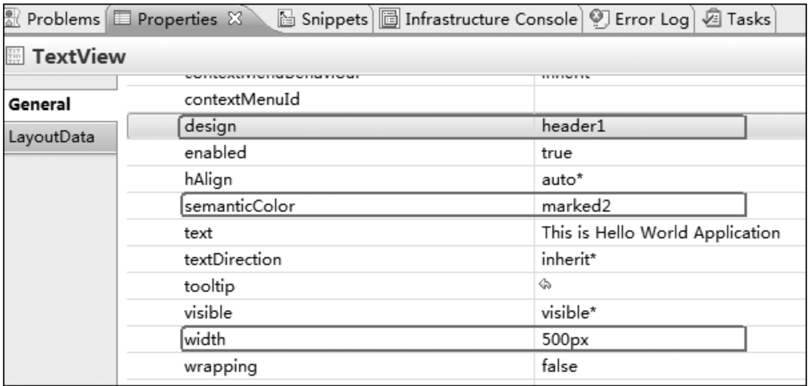


图 2-19　修改元素属性

2.3　配置服务器

1）在部署 Web Dynpro 程序之前，还需要配置部署服务器的地址，如同在运行 HTML 页面之前必须指定 tomcat 地址一样。单击“Window”→“Preferences”命令，进行配置管理，如图 2-20 所示。

2）在弹出的对话框中左上方的文本框中输入“sap”进行模糊搜索（见图 2-21），在左侧的列表中选择“SAP AS Java”，默认右侧列表内容是空的，单击右侧的“Add”按钮。

3）输入需要部署程序的服务器进行连接，如图 2-22 所示。

注：图 2-22 中的 IP 仅作为参考，请根据实际服务器地址进行相关配置。

4）完成添加后，就会出现对应服务器的主机名，这样就可指定服务器并部署 Web Dynpro 程序了，如图 2-23 所示。

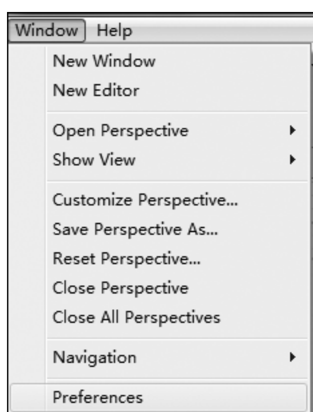


图 2-20 修改配置菜单

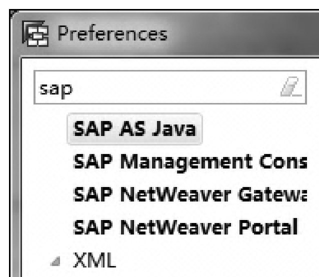


图 2-21 配置中的模糊查询

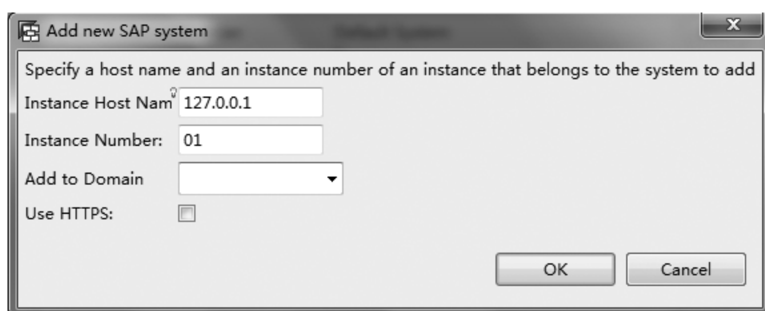


图 2-22 输入服务器连接信息

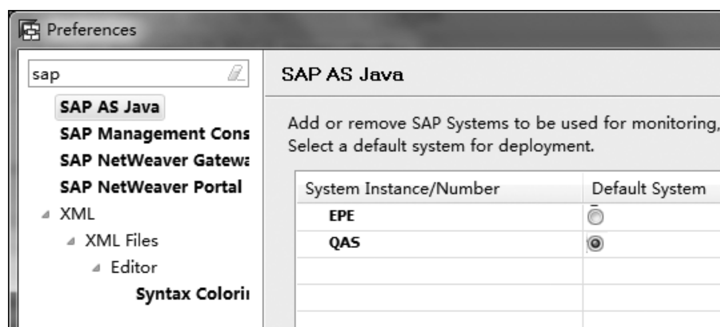


图 2-23 服务器信息配置后示例

2.4 部署应用程序

1）回到项目目录中，右击“Applications”，在弹出的快捷菜单中选择“Create Application”，创建一个应用程序，如图 2-24 所示 。

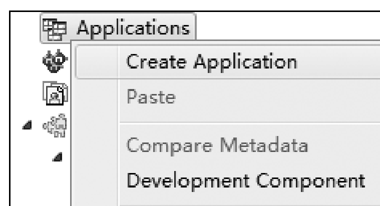


图 2-24 创建一个应用程序

2）输入应用程序的名称“Dynp01App”（见图 2-25），单击“Next”按钮。

3）选中“Use Existing Component”单选按钮（见

图 2-26），单击“Next”按钮，确认信息后，单击“Finish”按钮。

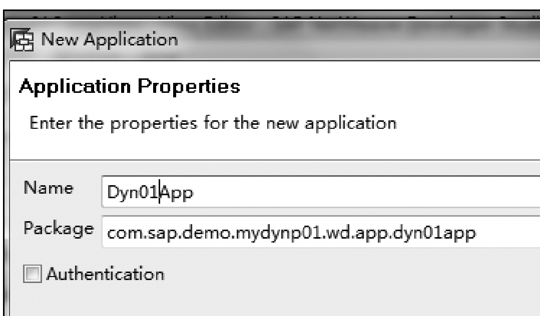


图 2-25　输入应用程序名称

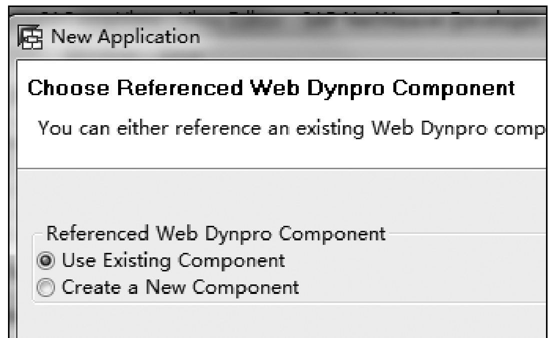


图 2-26　选择引用组件类型

4）完成应用程序的创建后，就可以看到在模型编辑器中应用程序其实是引用的组件中的窗口的接口视图，如图 2-27 所示。

5）右击新建的“Dynp01App”应用程序，在弹出的快捷菜单中选择“Deploy New Archive and Run”，部署新版本并运行，如图 2-28 所示。

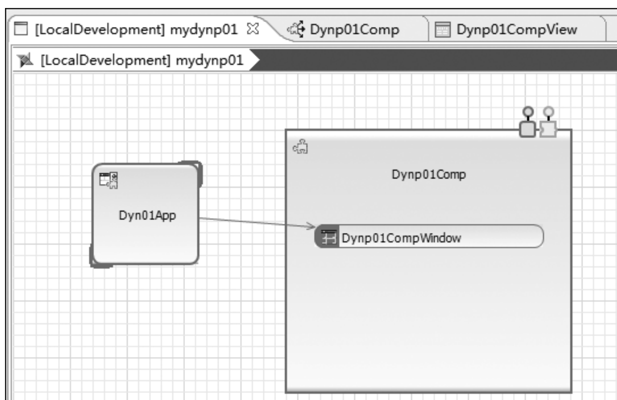


图 2-27　模型编辑器中的效果

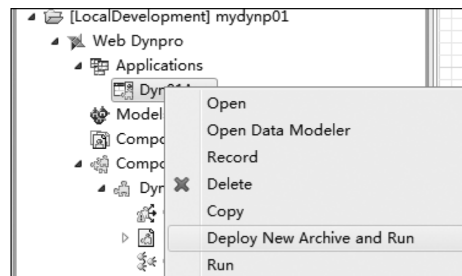


图 2-28　运行应用程序

6）部署完毕，弹出确认对话框，单击“OK”按钮（建议选中“Donot show this dialog again”复选框），如图 2-29 所示。

7）自动打开默认浏览器，并在指定的服务器端运行刚才实例中创建的“Hello World ”应用，运行效果如图 2-30 所示。

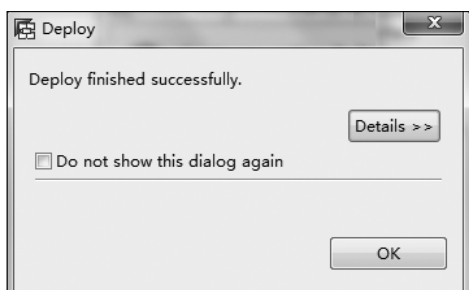


图 2-29　部署完毕的确认对话框



图 2-30　浏览器运行效果

第 3 章　WD4J 工程详解

本章将详细地对 Web Dynpro for Java 工程各个部分进行介绍，包括 Web Dynpro 的视图、窗口、控制器等内容，同时每一节的内容都会补充相关的开发实例，帮助读者更好地将理论与开发实践相结合。

3.1　WD4J 工程简介

SAP 对于标准的 MVC 设计模式做了一些重要的改变：

- 标准的 MVC 允许模型发生变化时直接通知对应视图，这并未在 Web Dynpro 中应用。
- 标准的 MVC 允许嵌套型视图控制器，这在 Web Dynpro 中是不允许的。
- SAP 对设计模式进行了扩展，添加了一个集合体，也就是 Web Dynpro 工程中的组件（Component）。图 3-1 所示是一个典型的 Web Dynpro 组件。

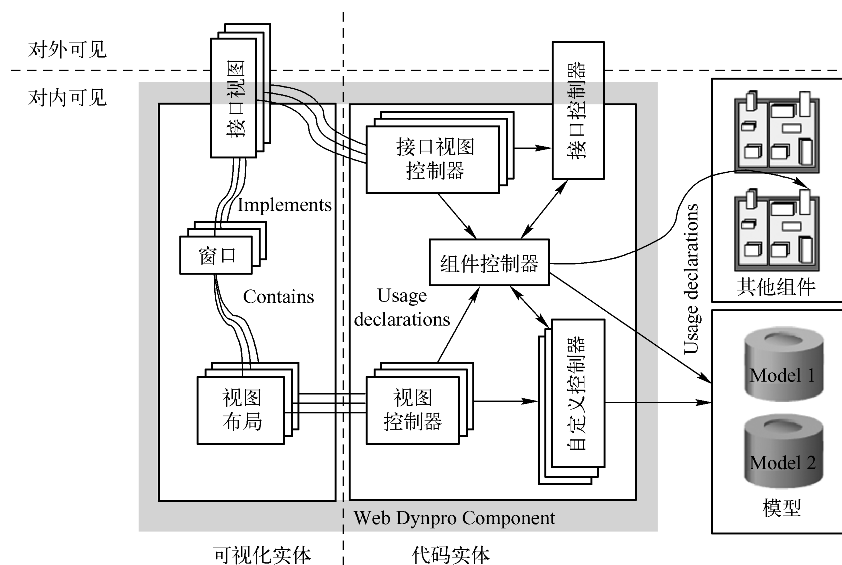


图 3-1　Web Dynpro 组件架构图

Web Dynpro 组件的说明如下：

从整个架构来看，一个 Web Dynpro 组件可以分为以下几个部分：

① 水平虚线将整个组件的实体分割为两块：对组件外部可见部分（包括接口视图、接口控制器）和仅对组件内部可见部分（包括视图、窗口、控制器等）。

② 垂直虚线也将整个组件的实体分割为两块：可视化实体和不可视实体（或代码

实体)。

一个 Web Dynpro 组件仅有接口视图和接口控制器对外可见:

1) 所有的 Web Dynpro 组件都有一个接口控制器。

2) 一个窗口与一个接口视图始终是一对一的关系(注意,是窗口而不是视图)。

3) 组件里不存在窗口是“匿名”的。

- 虽然在一个 Web Dynpro 组件中所有的控制器都是独立编写代码,但它们都共同扮演着 MVC 中“C”的角色,并且没有控制器是脱离组件而存在的。
- Web Dynpro 模型,也就是 MVC 中的“M”是在组件之外的,因为在 Web Dynpro 中模型可以被多个组件交叉使用。
- 一个 Web Dynpro 组件可以声明引用其他 Web Dynpro 组件。在声明之后,一个组件引用会在父组件中创建,而子组件中功能和方法的调用则是通过子组件的接口控制器进行的。

Web Dynpro 工程实例:

回到在上一章中创建的 Web Dynpro 工程,再次看一下在实际开发工具中的目录结构(见图 3-2):

最顶层是整个 Web Dynpro 工程的根目录结点 [LocalDevelopment] mydynp01,这也是一个可发布的单元。

下一层包含 Web Dynpro、Dictionaries 和 Resources(在旧版本中是 src)。

- Web Dynpro:包含所有的 Web Dynpro 程序、模型和组件。
- Dictionaries:对应于具体的数据层,建立与后台表结构相映射的结构以及固定数据的选择范围。
- Resources:包括 Web Dynpro 所有定制开发组件的描述文件、资源文件(本地的图片、HTML 静态页面等)以及 Java 代码源文件。

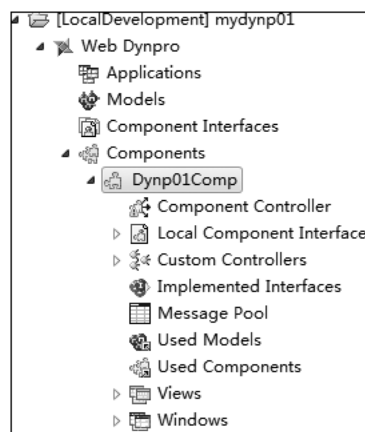


图 3-2 Web Dynpro 项目目录结构示例

在 Web Dynpro 中包含了应用程序(Applications)、模型(Models)、组件(Components)等部分。

- 应用:从开发人员的角度来看,应用是将组件封装打包后的可运行对象,是整个工程的“成品”;从最终用户的角度来看,程序同时创建了一个指向具体应用场景的用户入口,每个程序对应单独一个 URL 访问链接。
- 模型:用于封装外部业务数据模型,并创建唯一的接口作为远程服务调用,如 RFC、WebService 等外部系统接口调用。模型为组件提供了丰富的原始数据,即来自于不同系统或外部接口的“原料”。
- 组件:组件是整个 Web Dynpro 工程的核心,是工程的业务逻辑处理单元,包含了所有的可定制开发的应用场景,包括视图、窗口、组件控制器等对象。组件无法通过 URL 直接访问,但它的接口视图和接口控制器可以被其他组件所访问。

从逻辑角度来说,组件中的视图定义了用户的操作界面,窗口是视图的载体,同时一一对应地实现了一个接口视图。接口视图作为组件的出口,在创建应用时被引用。最终整个 Web Dynpro 工程建立了这样一个关联:View – Window – InterfaceView – Component – Applica-

tion，这也是第 2 章中 Hello World 工程的骨架。

3.2 视图

Web Dynpro 的视图（View）是唯一的 Web Dynpro 组件中与最终用户面对面交流的部分，用户的操作通过视图中的 UI 元素完成，而程序的逻辑处理则是通过视图的控制器来完成的。

3.2.1 视图的生命周期

当一个 Web Dynpro 组件实例创建时，默认视图也会创建相应的实例对象。如果不是组件的默认视图，则它会在指向它的导航请求触发时创建实例对象，直到窗口关闭或者有其他的页面切换请求被触发。

从 J2EE 的角度考虑，Web Dynpro 的视图组件开发可以看成是对 JSP 或 Servlet 的封装。该组件在服务端执行并生成可以在客户端浏览执行的 HTML 源文件，人们对视图的开发就演变成对 JSP 的高级开发，更多的图形界面元素是基于可定制与模板化，并伴随着代码的自定义生成。视图有其生命周期，视图中有一些方法在其生命周期的不同阶段点会执行。

整个视图生命周期中可能会执行到的标准方法如下。

- wdDoInit( )：这是所有控制器（包括组件控制器和视图控制器）的默认初始化方法，且在实例初始化时仅执行一次，因此对于视图中屏幕元素的初始化或者变量的默认值设定都在这个方法中完成。
- wdDoExit( )：与 wdDoInit( ) 相对，该方法在每个控制器结束时执行。视图的退出场景有很多，包括视图失效、窗口关闭、视图切换等。该方法也是每个控制器实例的出口。
- wdDoModifyView( )：该方法在视图实例初始化以及运行中视图的页面 UI 元素产生事件触发时执行。由于这个方法与视图页面 UI 元素直接相关，因此仅在视图控制器（View 的 Java 代码）中存在，组件控制器中没有这个方法。该方法的默认参数为 firstTime，布尔型，用于标记屏幕是第一次创建还是属于重绘；如需求有动态绘制 UI 元素等内容，创建动态元素的代码就在 wdDoModifyView 方法中编写，也可根据情况编写部分初始化代码。需要注意的是，由于每次 UI 控件事件的触发都会执行该方法，该方法在视图的整个生命周期中执行频率最高，因此除了表格的过滤与排序等必要的需求场景外，并不建议在 wdDoModifyView( ) 中编写过多代码。为了避免破坏 MVC 的设计理念，SAP 也将这个方法限制为静态类型。
- wdDoBeforeAction( )：该方法在界面控件的 Action 事件触发前执行。一般情况下，数据有效性校验会在 wdDoBeforeAction 中编写，校验会直接指向页面元素指针，若校验通过 Action，则继续执行；若校验失败，则抛出异常并且页面元素显示红色，程序中断。

项目开发人员需要特别注意方法执行顺序：

在视图初始化时，先执行 wdDoInit( ) 方法，后执行 wdDoModifyView( ) 方法。

在触发视图的界面 UI 控件的事件后，先执行 wdDoBeforeAction( )，然后执行控件事件，

最后执行 wdDoModifyView( )方法。

开发实例：

1）从刚刚新建的视图切换到 Java 编辑器视图中，选择“Outline”选项卡中的“wdDoInit( )”初始化方法，如图 3-3 所示。

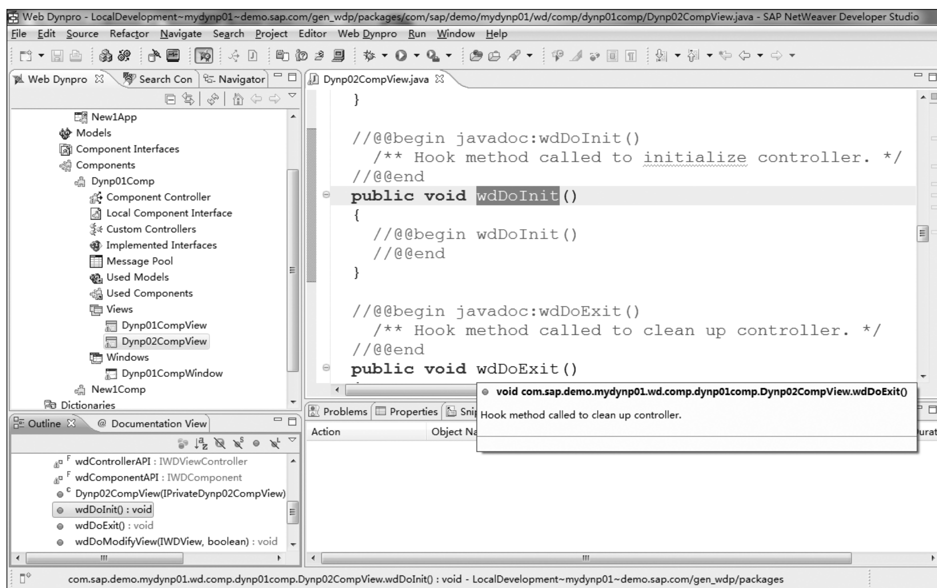


图 3-3 Java 编辑器中的初始化方法

2）在 wdDoInit( )方法中加入以下代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess( "Init" ) ;
```

3）在 wdDoExit( )方法中加入以下代码：

```
logger. infoT( "Exit" ) ;
```

4）在 wdDoModifyView 中加入以下代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess( "modifyView" ) ;
```

5）在 wdDoBeforeAction 中加入以下代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess( "BeforeAction" ) ;
```

6）完成代码工作后，需要对程序进行编译。右击“Web Dynpro”，在弹出的快捷菜单中选择“Development Component”→“Build”命令，如图 3-4 所示。

7）部署：右击“Web Dynpro”，在弹出的快捷菜单中选择“Development Component”→“Deploy”命令，如图 3-5 所示。

8）接着运行部署后的应用程序，右击“Applications”中的应用程序，在弹出的快捷菜单中选择“Run”，如图 3-6 所示。由于一个 Web Dynpro 工程允许存在一个或多个程序，因此在运行时需正确选择程序，同样在门户创建页面时也需要注意。

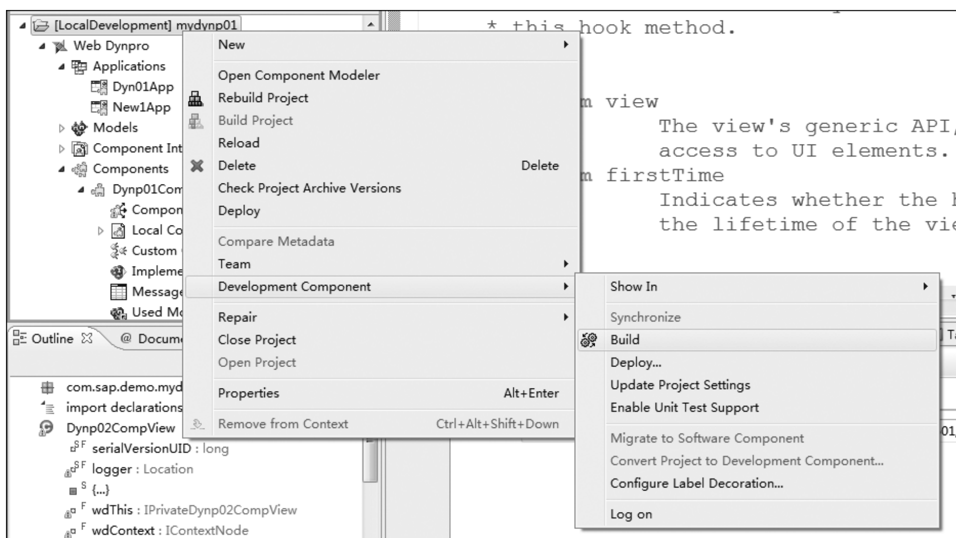


图 3-4　编译 Web Dynpro 项目

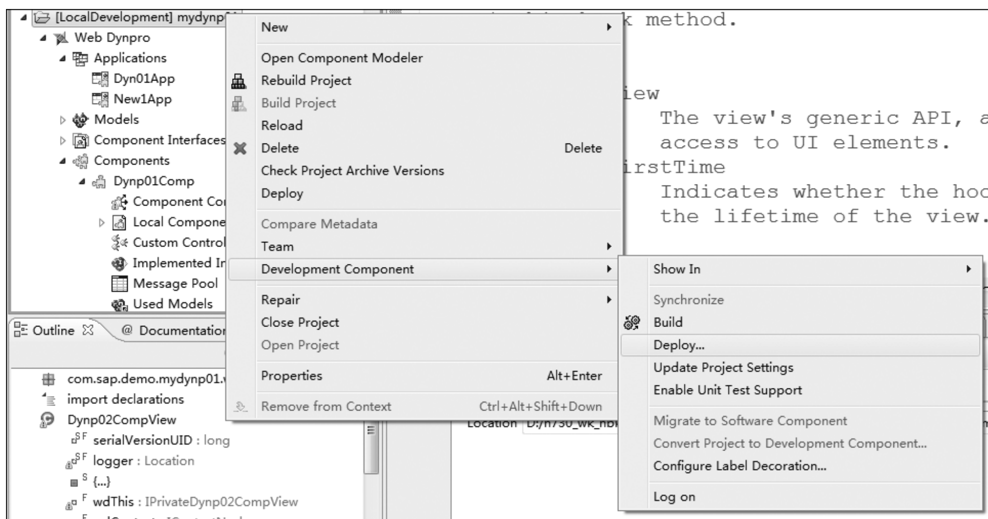


图 3-5　部署 Web Dynpro 项目

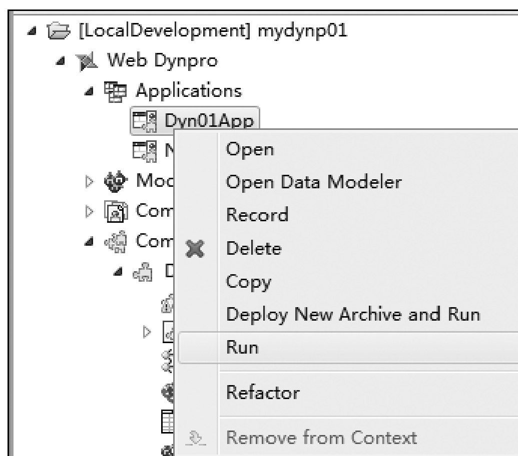


图 3-6　运行应用程序

9）运行后，程序将在浏览器中自动打开，上方是在 Web Dynpro 工程视图中创建的按钮，下方是默认的消息区域，如图 3-7 所示。

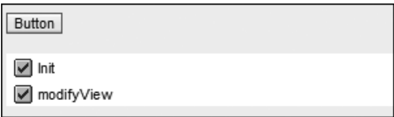


图 3-7　运行后的效果

10）单击“Button”按钮后的效果如图 3-8 所示。



图 3-8　单击“Button”按钮后的效果

从执行的结果可以看出，初始化先打印了 wdDoInit 方法中的消息内容，然后打印了 wdDoModifyView 方法中的消息内容。当触发 Button Action 时（即单击按钮时），先打印 wdDoBeforeAction 方法中的消息内容，然后打印 Action 方法中的消息内容，最后打印 wdDoModifyView 方法中的消息内容。

3.2.2　视图的描述文件与标准类

在 Web Dynpro 中，每次创建一个视图都会默认生成 4 个描述文件与 3 个类文件。图 3-9 所示为对应自动生成的类文件，以及相互之间的继承关系，所有自定义代码仅在视图控制器中完成，其余的接口、实现类都会自动生成。其他相关文件还包含所有页面元素的描述文件、视图控制器文件，以及多语言配置的 . xlf 文件等。

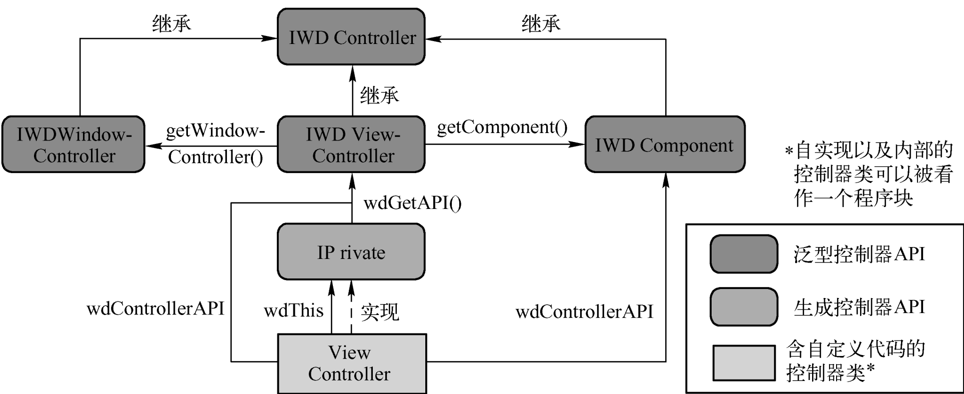


图 3-9　视图相关类以及相互之间的继承关系

确切地说，开发工具在创建一个视图之后会在文件目录中自动产生 7 个文件，开发人员或运维人员对视图的任何编辑即为对下面几个文件的联动修改，而所有视图的页面元素和控制逻辑都包含在对应的文件中。

- . view：视图页面元素对应的描述记录文件。
- . controller：视图控制器对应的描述记录文件。
- . xlf：用于多语言记录的语言属性文件，通过对该文件的多语言复制，可以实现多语言的场景应用。
- <NAME> View. java：视图对应的后台处理类，包含视图事件处理，用户交互所包含的业务处理逻辑，开发及运维人员重点关注的文件。
- IPrivate <NAME> View. java：视图中 wdThis 对应的实例接口，包含所有的视图相关 API 以及数据结构定义，代码自动生成。
- Internal <NAME> View. java：实现 IPrivate <NAME> View 接口，视图中 wdThis 所指向的对象实例，代码自动生成。

开发实例：

1）如果需要查看视图的描述文件，则进入 NWDS 的 Navigator 视图（也可以通过 Web Dynpro 视图展开 resources 目录），并找到对应工程的文件夹，如图 3-10 所示。

2）在“src”文件夹下，找到对应的“Component”文件夹，文件夹名字即之前创建的组件的名字，如图 3-11 所示。

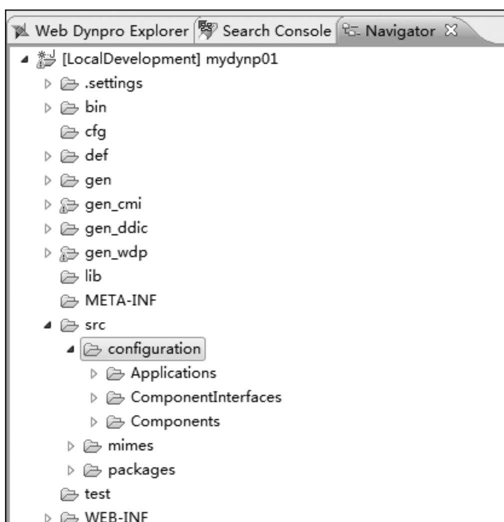


图 3-10　导航器中的文件目录

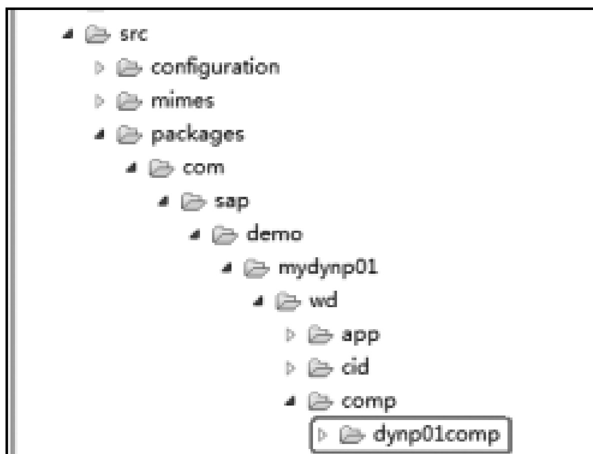


图 3-11　组件对应的文件目录

3）展开“comp”文件夹，找到新建的视图所对应的几个文件，如图 3-12 所示。

注：*在极少的情况下，需要通过这种方式直接修改对应描述文件。*

4）在实际的 Web Dynpro 工程的目录中，通过 Navigator 定位到“gen_wdp”目录，如图 3-13 所示。

5）展开“packages”，定位到“comp”下的视图，双击图中标注的“Dynpro02Comp View”类，（见图 3-14），可以发现该类文件与双击“Button”按钮后打开的类文件为同一文件。该文件即为视图对应的后台 Java 处理类。该类中包含了所有视图中自定义的业务逻辑单元。

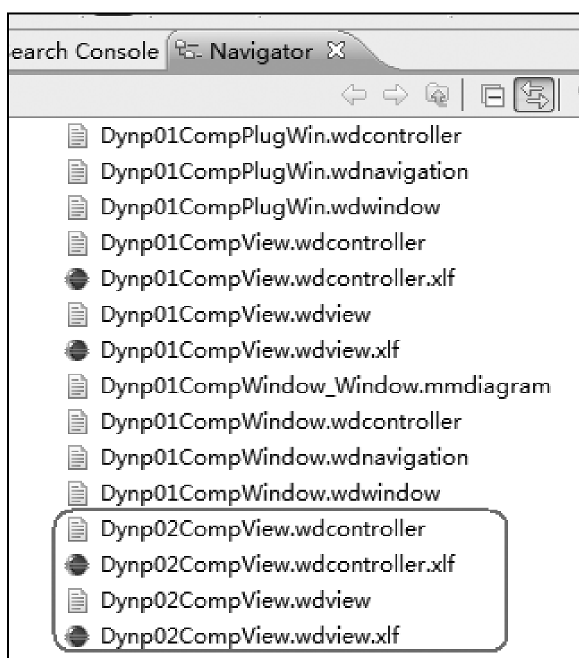


图 3-12　视图对应的实际文件

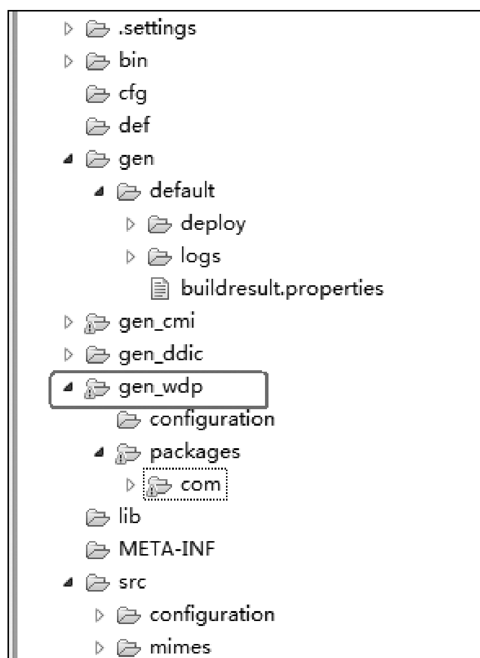


图 3-13　选择“gen_wdp”目录

6）展开“wdp”目录，定位到对应视图的类文件。由于在之前实例中创建的视图名称是 Dynp02CompView，因此自动产生的 IPrivateDynp02CompView 为相应视图控制器中的 wdThis 实例接口，InternalDynp02CompView 为外部使用的类文件，如图 3-15 所示。

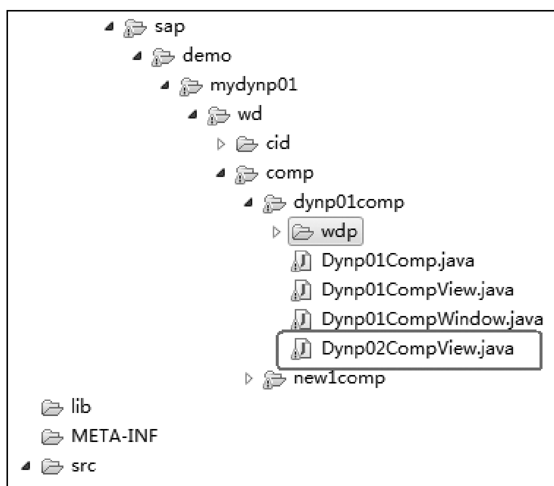


图 3-14　视图类文件的目录位置

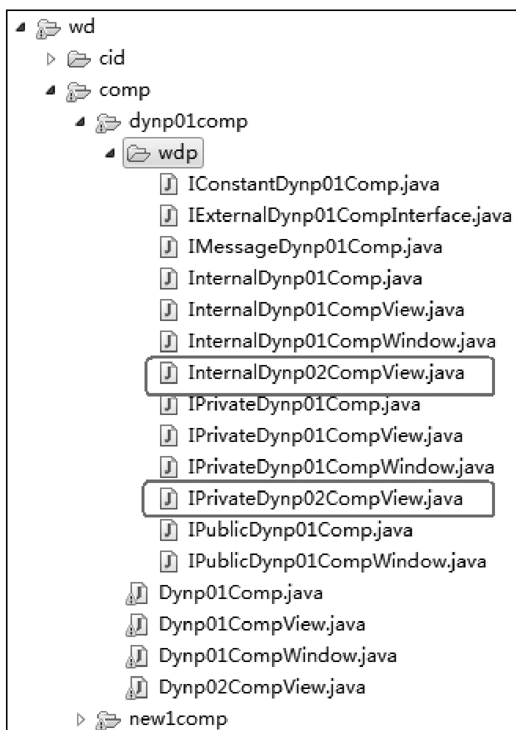


图 3-15　相关类文件的目录位置

3.2.3 视图的上下文

1. 上下文的定义

Web Dynpro 中的 Context 即程序的上下文。在 Web Dynpro 工程中，除了接口视图控制器以外的所有控制器和视图都有一个存储数据的多级结构，而这就是 Web Dynpro 中的上下文。

从宏观对象的角度理解，上下文可以看作视图对象对应的逻辑属性，或者是视图对象的环境变量，所有前端页面的数据交互都会通过上下文传递到对应的控制器中进行逻辑处理。另外，它还有以下几个特点：

- 所有上下文中的数据仅存在于控制器的实例中，一旦实例运行结束，上下文中的数据也将会被释放。
- 上下文的结构是在开发阶段定义的，程序运行时无法对其进行任何更改。
- 视图的上下文是私有的，但组件控制器的上下文可以通过对外声明进行公开，也可以进行映射，如控制器与视图的上下文映射，控制器与模型的上下文映射等，这也是 Web Dynpro 中严格进行 MVC 架构的一个体现。

从底层代码的分析可以发现，上下文源自 IPrivate <NAME> View 接口的静态内部类，所有在可视化界面中对上下文的操作都会间接地去修改该静态类，并动态地生成对应该上下文数据结构的操作方法。假设创建了一个视图，则它的上下文来自于 IPrivate <NAME> View 接口中的 IContextNode 静态类。下面通过一个开发实例进行进一步讲解和验证。

开发实例：

1）切换到新建的视图，在“Properties”中单击“Context”，如图 3-16 所示。

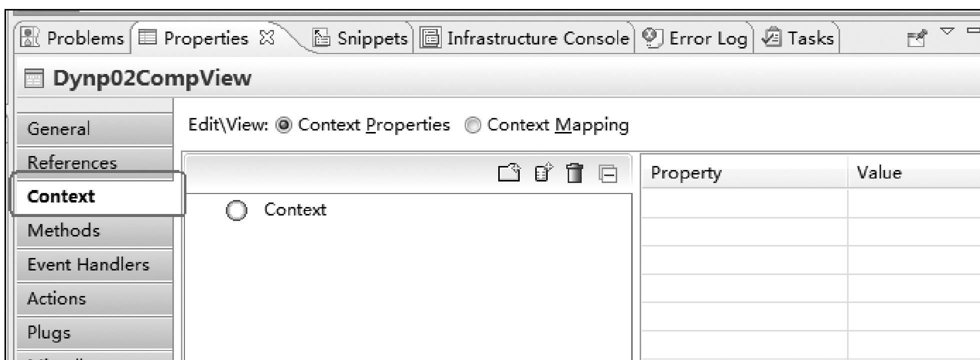


图 3-16 单击“Context”

注：在 NetWeaver 7.0 及其之前版本的开发工具中，上下文 Context 的定义在与布局“Layout”并列的“Context”中，然而在新版本的开发工具中，上下文、方法、事件处理器等内容统一被包含在“Properties”选项卡下。

2）右击“Context”，在弹出的快捷菜单中选择“New”→“Attribute”命令，创建新特性，如图 3-17 所示。

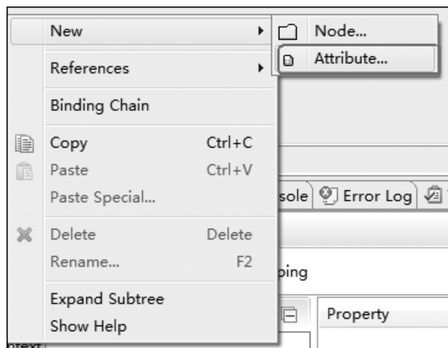


图 3-17 创建新特性

3）选择自定义创建，然后输入特性名称并选择特性的类型。选中“Manually”单选按钮进行手

动创建，并输入“name”，选择“string”字符串类型，如图 3-18 所示。

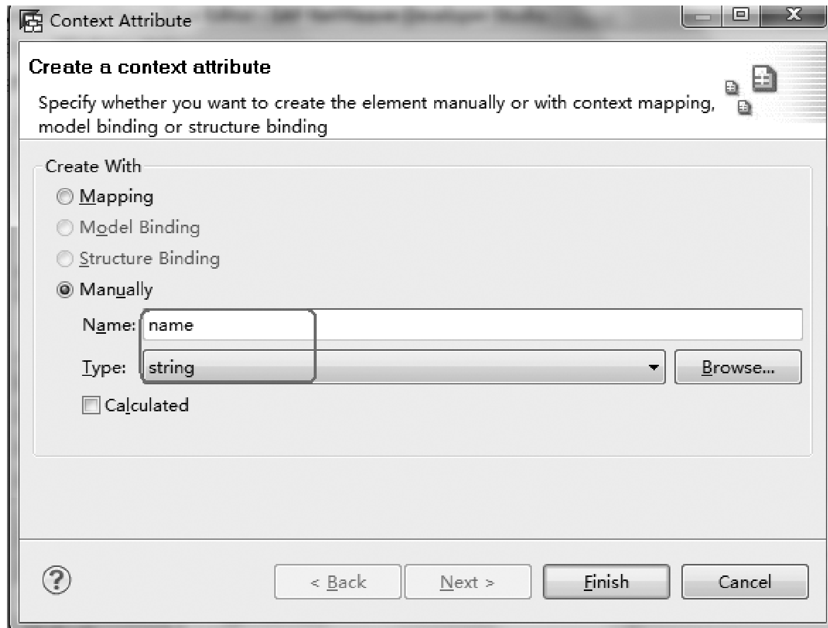


图 3-18　输入特性名称并选择特性类型

4）验证上下文在 Java 源文件中定义。切换到 IPrivateDynp02CompView 类，通过 Outline 方法清单或按〈Ctrl + O〉组合键找到 IContextElement 静态类，可以看到刚才新建的属性对应的静态对象声明（见图 3-19）以及 get 和 set 赋值方法都已自动创建完毕（见图 3-20）。

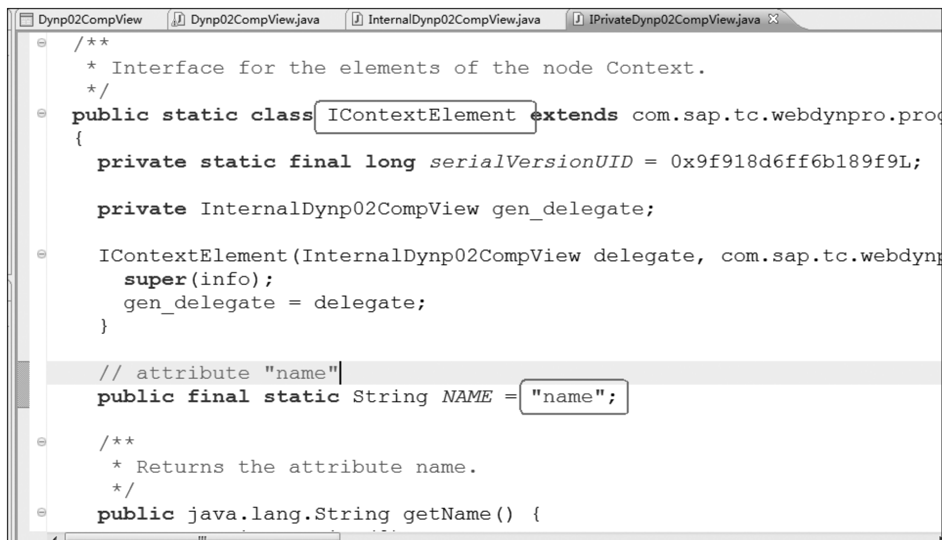


图 3-19　静态对象声明

5）通过代码对 Context 进行赋值，右击对应的视图，选择打开 Java 编辑器，如图 3-12 所示。

```
public final static String NAME = "name";

/**
 * Returns the attribute name.
 */
public java.lang.String getName() {
  return gciGetString(0);
}

/**
 * Sets the attribute name.
 * @param value the new value
 */
public void setName(java.lang.String value) {
  gciSetString(0, value);
}
```

图 3-20　自动创建的方法

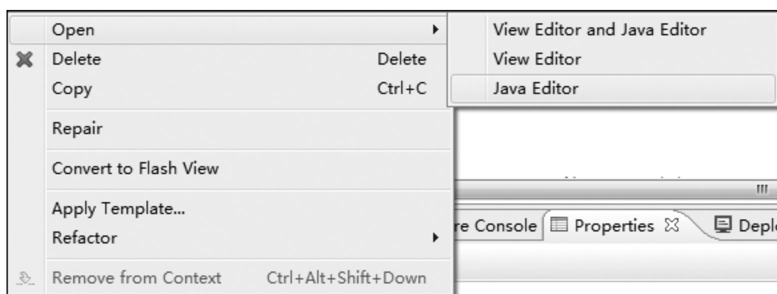


图 3-21　打开 Java 编辑器

6）找到视图初始化的默认“wdDoInit”方法，并加入以下赋值代码：

```
public void wdDoInit( )
{
    //@ @ begin wdDoInit( )
        //将默认 Context 元素中的 name 属性的值设为“王明”
        wdContext. currentContextElement( ). setName( " 王明" ) ;
    //@ @ end
}
```

7）找到“onActionButton”方法，给之前的代码加上注释并加入以下代码：

```
//将默认 Context 元素中的 name 属性的值取出并赋给 name 变量
String name = wdContext. currentContextElement( ). getName( ) ;
//将 name 变量以成功消息形式打印出来
wdComponentAPI. getMessageManager( ). reportSuccess( name) ;
```

8）保存所有修改，重新编译运行之后，即可进行相关测试。单击“Button”按钮可以看到打印出的“name”特性的信息，如图 3-22 所示。由于是成功类型的消息，因此文本前有一个绿色的感叹号。

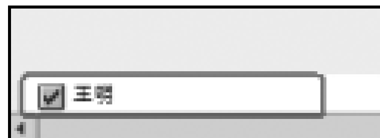


图 3-22　程序运行效果

2. 上下文的属性

由于上下文是可分级的数据，因此不管是上下文的结点或者特性，都可以对它们的属性进行进一步的配置。下面通过上一节创建的实例进行进一步说明。

（1）特性的属性

打开页面的“属性”选项卡，找到刚才创建的 name 特性，单击选中后，可以看到右边还有相关的属性可以进行配置，如图 3-23 所示特性的属性名称及说明见表 3-1。

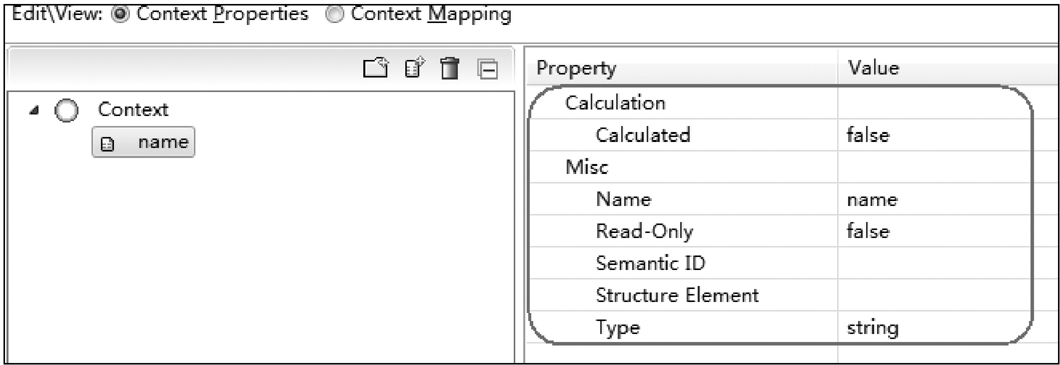


图 3-23　特性的属性配置

表 3-1　特性的属性名称及说明

属 性 名 称	说　　明
Calculated	是否可计算（如为 true，将自动生成对应的计算方法对特性进行初始化）
Name	特性的名称
Read - Only	是否只读，该属性与 Calculated 属性相关，设置为 true 时，Calculated 属性没有 setter 方法
Semantic ID	语义 ID，很少使用
Structure Element	对应的特定结构的数据类型，很少使用
Type	类型，可通过下拉菜单选择简单类型（如 string、date、int 等基本类型），也可通过单元格右侧的按钮选择其他复杂类型（包括本地数据字典中的自定义类型）

（2）结点的属性

上下文 Context 中的结点是存放 Web Dynpro 数据的一个虚拟类，它允许有子结点（包含了结点和特性两种类型），所以它也可以看作一个集合，可以想象是一个表格或者树的根结点。结点属性名称及说明见表 3-2。

表 3-2　结点的属性名称及说明

属 性 名 称	说　　明
Collection Cardinality	结点的集合属性
Initialize lead Selection	是否初始化首选项（如在表格中有默认选中行）
Name	结点的名称
Selection Cardinality	选择的集合属性
Singleton	是否单例（通用设计模式的一种）
Structure	结点的创建可以基于任意数据结构，对于通过模型绑定或者自定义的结构类型建立的结点，该处显示结构名称
SupplyFunction	提供对结点填充的处理方法

下面对结点的集合属性和选择的集合属性中的下拉列表的选项进行说明。

1）Collection Cardinality 中：

① 0..1 代表 node 集合长度为 1，默认集合为空，可以添加一个实例。

② 0.. n 代表 node 集合长度为 n，默认集合为空，可以添加 n 个实例。
③ 1.. 1 代表 node 集合长度为 1，初始化默认一个实例，不可以再次添加实例。
④ 1.. n 代表 node 集合长度为 n，初始化默认一个实例，可以添加 n 个实例。
2）Selection Cardinality 中：
① 0.. 1 代表最多一条的选中，默认没有选中条目。
② 0.. n 代表最多 n 条的选中，默认没有选中条目。
③ 1.. 1 代表最多一条的选中，默认一条选中。
④ 1.. n 代表最多 n 条的选中，默认 n 条选中。

开发实例：

1）右击 Context，在弹出的快捷菜单中选择“New”→“Node”命令，新建一个结点，如图 3-24 所示。

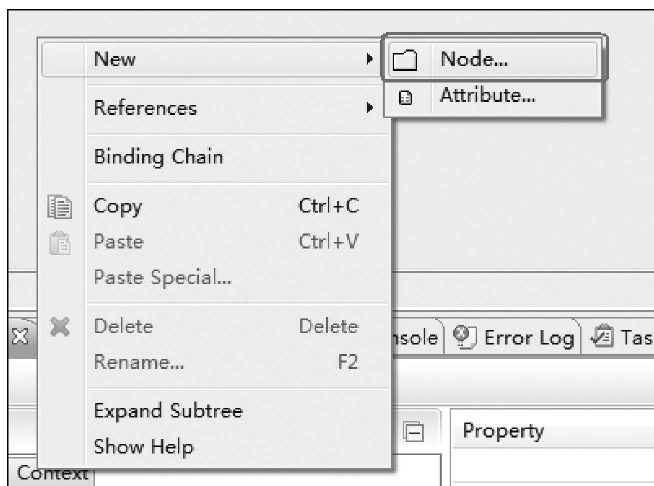


图 3-24　新建一个结点

2）输入结点名称“Per”，单击“Finish”按钮，如图 3-25 所示。

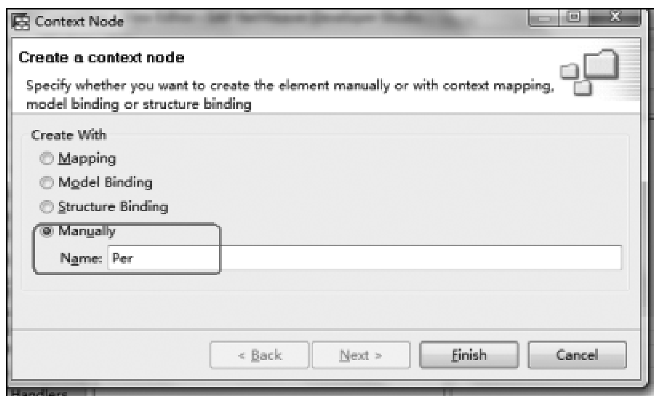


图 3-25　输入结点名称

3）接着验证 Context 在 Java 文件中的定义：在保存好上述修改后，切换到“Internal Dynp02CompView”类，可以看到静态语句块中自动生成了对刚才新建 node 的初始化代码如图 3-26 所示。其中参数中的 CMICardinality 分别对应于结点的集合属性和可选属性。

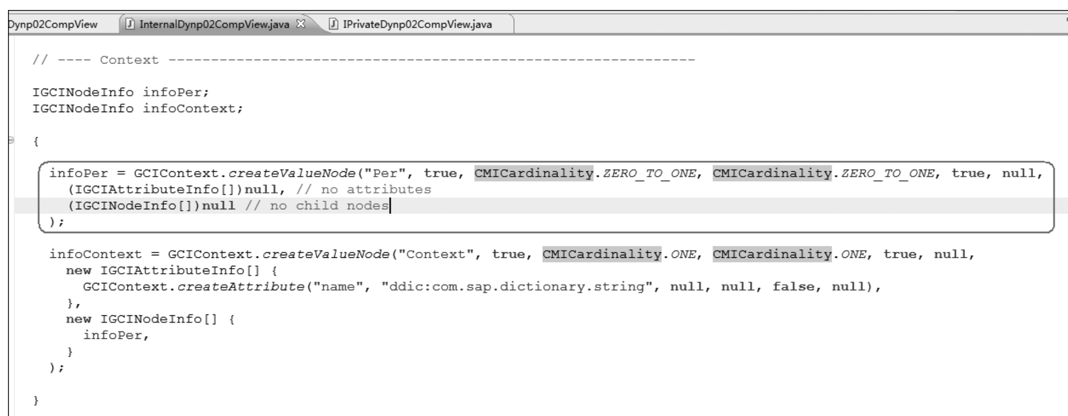


图 3-26　自动生成代码定义

4）打开页面的“属性”选项卡，找到刚才创建的“Per”结点，单击选中后，可以看到右边还有相关的属性可以进行配置，如图 3-27 所示。

5）通过代码对结点中的特性进行赋值。在“Per”结点中新建一个名称为“id”的特性，如图 3-28 所示。

Property	Value
Documentation	
Quick info	
Technical Documentation	
Misc	
Collection Cardinality	0..n
Initialize Lead Selection	true
Name	Per
Selection Cardinality	0..1
Singleton	true
Structure	
Supply Function	
Typed Access Required	true

图 3-27　结点的属性

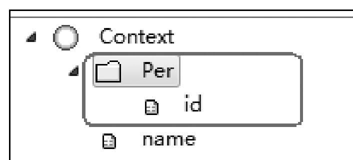


图 3-28　新建一个名称为“id”的特性

6）作为测试用例，将“Per”结点的集合属性修改为“0..1”，如图 3-29 所示。

Edit\View: Context Properties　Context Mapping

Context
Per
id
name

Property	Value
Documentation	
Quick info	
Technical Documentation	
Misc	
Collection Cardinality	0..1
Initialize Lead Selection	false
Name	Per
Selection Cardinality	0..1
Singleton	true
Structure	
Supply Function	
Typed Access Required	true

图 3-29　结点属性示例

7）切换到视图的“wdDoInit”初始化方法，加入以下代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess ( " PerNode Size:" + wdContext. nodePer ( )
. size( ) );//打印结点的大小
IPerElement createAndAddPerElement = wdContext. nodePer ( ). createAndAddPerElement ( );//添
加 Element
createAndAddPerElement. setId( "12" );
wdComponentAPI. getMessageManager( ). reportSuccess ( " PerNode Size:" + wdContext. nodePer ( )
. size( ) );//打印添加 Element 后 node size
```

8）进入“Button”按钮的动作处理逻辑中，并加入以下代码：

```
messageManager. reportSuccess( wdContext. nodePer( ). currentPerElement( ). getId( ) );
//打印当前选中(默认选中)Element 的 Id 变量值
```

9）与之前同样地进行编译部署，在运行之后，单击“Button”按钮，如图 3-30 所示。

10）接着单击“Button”按钮，打印出添加的特性值。



图 3-30　运行效果

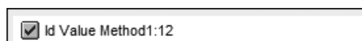


图 3-31　单击按钮后的效果

（3）特性的可计算属性

特性的属性中的 Calculated 比较特别，除了 true 或者 false 的选择外，还涉及代码编写，所以在此单独进行说明与开发实例展示。该属性设置为 true 时，代表可以自定义特性值的获取逻辑，从字面理解来看也就是可以计算的意思，并且在对应的类中会生成对应的 getter 方法。

开发实例：确认自动生成的公共方法及其调用逻辑

1）切换到视图的上下文界面，将“Per”结点下的“id”特性的 Calculated 属性设置为“true”，如图 3-32 所示。根据上下文类定义的介绍，依次查看视图控制器 Dynp02Comp View. java、公共接口 InternalDynp02CompView. java 和实现类 IPrivateDynp02 CompView. java 3 个 java 文件中自动生成的代码。

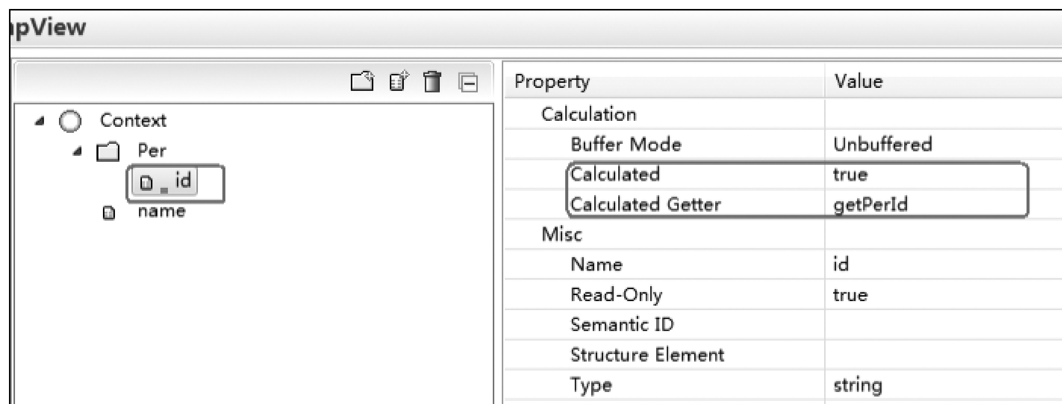


图 3-32　特性的属性

2）单击“Calculated Getter”中“getPerId”最右侧的“Go”按钮，界面将自动跳转到Dynp02CompView. java 中生成的 getPerId( )公共方法（见图 3-33），传入参数 element 是“Per”结点的实例元素。

```
Dynp02CompView | Dynp02CompView.java | InternalDynp02CompView.java | IPrivateDynp02CompView.java
    //@@begin javadoc:getPerId
    /**
     * Declared getter method for attribute id of node Per.
     *
     * @param element the element requested for the value
     * @return the calculated value for attribute id
     */
    //@@end
    public java.lang.String getPerId(IPrivateDynp02CompView.IPerElement element)
    {
      //@@begin getPerId
      return null;
      //@@end
    }
```

图 3-33　自动生成的代码 1

3）进入 InternalDynp02CompView. java，同样看到生成了 getPerId( )方法（见图 3-34），内部实现为调用 Dynp02CompView 实例的方法。

```
2CompView | Dynp02CompView.java | InternalDynp02CompView.java | IPrivateDynp02CompView.java
  public void method_Fill_DropDownWithProposalList( com.sap.typeservices.IProposalList
    delegate.method_Fill_DropDownWithProposalList( _proposalList);
  }

  public void method_Fill_FieldProposalList_1( ) {
    delegate.method_Fill_FieldProposalList_1();
  }

  public java.lang.String getPerId(IPrivateDynp02CompView.IPerElement element) {
    java.lang.String result = delegate.getPerId(element);
    return result;
  }
```

图 3-34　自动生成的代码 2

4）进入 IPrivateDynp02CompView. java，定位到 IPerElement 内部类，看到 wdGetCalculated( )方法中添加了执行 InternalDynp02CompView 实例中 getPerId( )方法的代码，如图 3-35 所示。由于 InternalDynp02CompView 中 getPerId( )方法执行 Dynp02CompView 中 getPerId( )逻辑，因此最终是执行了视图控制器中的自定义代码。

```
CompView | Dynp02CompView.java | InternalDynp02CompView.java | IPrivateDynp02CompView.java
     * @param value the new value
     */
    public void setId(java.lang.String value) {
      gciSetString(0, value);
    }


    protected Object wdGetCalculated(IGCIAttributeInfo info) {
      switch (info.getIndex()) {
        case 0: // id
          return gen_delegate.getPerId(this);
      }
      return null;
    }
}
```

图 3-35　自动生成的代码 3

开发人员一般关注视图控制器的 java 文件即可，如 Dynp02CompView. java，而接口类和实现类的代码都会自动生成。

开发实例：验证 getter 方法的执行时间与执行顺序

1）切换到 Dynp02CompView 类中的 getPerId( )方法（见图 3-36），并添加以下代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess( "Execute getPerId Method" );
```

Dynp02CompView | Dynp02CompView.java | InternalDynp02CompView.java | IPrivateDynp02CompView.java

```
//@@begin javadoc:getPerId
/**
 * Declared getter method for attribute id of node Per.
 *
 * @param element
 *            the element requested for the value
 * @return the calculated value for attribute id
 */
//@@end
public java.lang.String getPerId(IPrivateDynp02CompView.IPerElement element)
{
  //@@begin getPerId
    wdComponentAPI.getMessageManager().reportSuccess(
            "Execute getPerId Method");
    return null;
  //@@end
}
```

图 3-36　添加自定义代码

2）修改 node Per 的集合属性为 1.. 1、Initialize Lead Selection 为 true（见图 3-37），同时注释掉 wdDoInit( )中的代码，进入 Button Action 并修改为以下代码：

```
messageManager. reportSuccess( "Id Value Method1 :" +
wdContext. nodePer( ). currentPerElement( ). getId( ) );
```

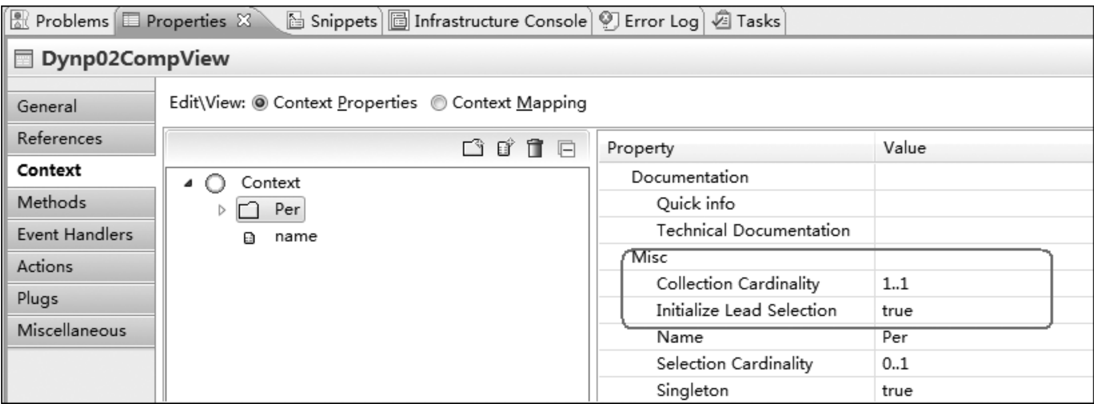


图 3-37　结点属性设置

3）保存所有修改，并重新编译、运行之后，即可进行相关测试，运行效果如图 3-38 所示。

单击“Button”按钮，可以看到先打印 Execute 语句后打印 Id Value，系统在执行 current * Element. getId( )方法前，执行了 getPerId( )方法，并且最终 Id 的数据源来自 getPerId 代码逻辑。由于 getPerId( )返回为 null，因此打印的 Id 值为 null。

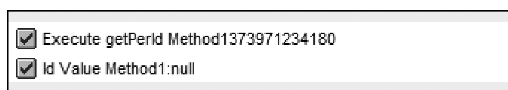


图 3-38　运行效果 1

4）修改 getPerId( )代码逻辑，保存所有修改，并重新编译，运行之后，即可进行相关测试。

```
wdComponentAPI. getMessageManager( ). reportSuccess( "Execute
  getPerId Method" + System. currentTimeMillis( ) );
return "element Of Id Attribute:" + element + ":abc";
```

5）单击“Button”按钮，程序返回了 Id 所属的当前 Element 索引以及自定义的拼接字符串，如图 3-39 所示。

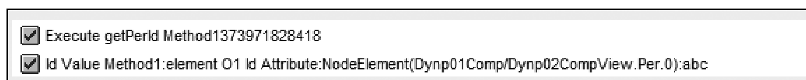


图 3-39　运行效果 2

3. 上下文的相关 API 及常用方法

1. IContextNode 上下文默认结点相关公共方法

切换到 IPrivate < NAME > View，定位到内部类 IContextNode 的公共方法，如图 3-40 所示。

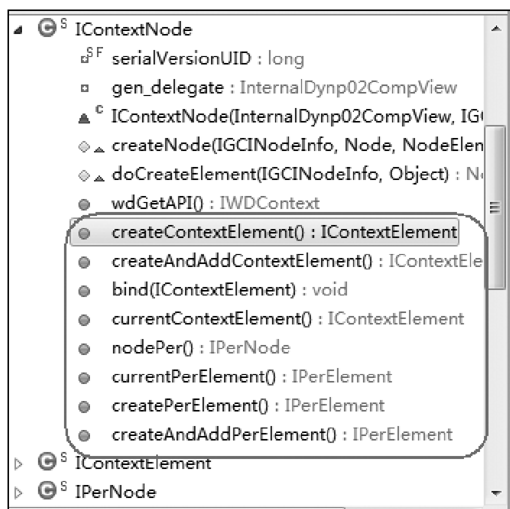


图 3-40　定位到内部类 IContextNode 的公共方法

1）**createContextElement( )**：创建一个该结点类型的上下文元素（element），但结点的索引并不能指向该元素的引用，即无法在结点中找到该元素，需要在代码中另外对结点的实例执行 add 方法进行添加，所以一般使用下面的 createAndAdd 方法。

2）**createAndAddContextElement( )**：创建一个上下文元素并添加至对应的结点实例中，即结点的集合属性的大小增加了，并且可以通过 getElement( )方法或 currentElement 方法（当前元素）获取该条 Element 的索引。注：集合属性为“1..1”的结点不允许执行 createAndAddElment( )方法，因为这样的结点有且仅有一个元素。

3）**currentContextElement( )**：获取当前首选（Lead Selection）的元素，只有结点存在首选元素时，该方法才能获取对应的元素，以下为 currentContextElement 在不同属性下的获取值情况：

当 Initial Lead Selection 为 false 时，并且结点在没有 Lead Selected 的情况下，currentContextElement 始终返回为空。

当 Initial Lead Selection 为 false 时，并且结点在有 Lead Selected 的情况下，currentContextElement 返回当前已选中的元素。

当 Initial Lead Selection 为 true 时，并且结点在没有 Lead Selected 的情况下，始终返回第一条元素。

Node <NAME>( )：获取 <NAME> 名称的子结点的引用。

current <NAME> Element( )：返回 <NAME> 名称的子结点对应的当前元素。

create <NAME> Element( )：创建一个 <NAME> 名称的子结点的元素。

createAndAdd <NAME> Element( )：创建一个 <NAME> 名称的子结点的元素，并将引用添加至子结点中。

2. IContextElement 上下文默认结点元素的相关公共方法

1）**get <NAME>( )**：IContextElement 下定义的特性 get 方法，获取 <NAME> 名称的特性值。

2）**set <NAME>( )**：IContextElement 下定义的特性 set 方法，设置 <NAME> 名称的特性值。

3. I <NAME> Node 自定义结点的相关公共方法

1）**current <NAME> Element( )**：同 IContextNode，返回 <NAME> 名称的子结点对应的当前元素。

2）**create <NAME> Element( )**：同 IContextNode，创建一个 <NAME> 名称的子结点的元素。

3）**createAndAdd <NAME> Element( )**：同 IContextNode，创建一个 <NAME> 名称的子结点的元素，并将引用添加至子结点中。

4）**get <NAME> ElementAt( )**：该方法针对集合属性为 0.. n 或 1.. n 的结点，获取指定索引的元素。

4. I <NAME> Element 自定义 <NAME> 结点元素（可以看作 <NAME> 结点的实例）的相关公共方法（见图 3-41）

1）**get <NAME>( )**：同 IContextElement，获取 <NAME> 名称的特性值。

2）**set <NAME>( )**：同 IContextElement，设置 <NAME> 名称的特性值。

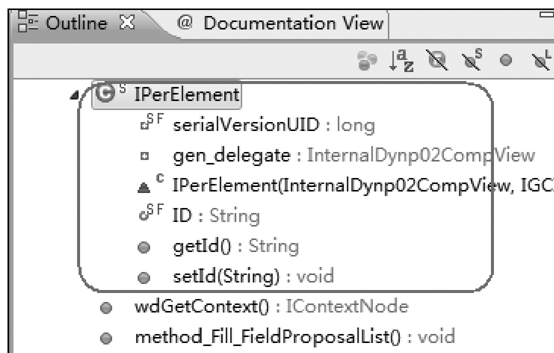


图 3-41　结点元素相关公共方法

以上整体介绍了默认上下文的 IContextNode、IContextElement 以及自定义结点的 I <NAME> Node、I <NAME> Element 的相关 API 的说明。而从 IContextNode 的初始化方法可以看到，该 Node 其实是集合属性为 1.. 1，并且选择属性也为 1.. 1 的结点，因此通常把它叫作根结点，可以在根结点的基础上去构造符合个性化需求的结构化上下文，并且这些数据结构会在 IPrivate <NAME> View 的内部类中自动生成对应的公共方法。当然这些新的内部类以及类方法同样遵循上述的规则，并以此类推控制整个上下文及其结构。

从 Web Dynpro 开发组件的角度来理解，Context 是具体某一 DC 组件的上下文，通过对

上下文的操作来实现对整个组件的属性控制。上下文中所有结点结构都对应于后台的内部类，并且针对内部类有对应的构造与操作变量的方法。

开发实例：

（1） 上下文属性实例 1

1） 修改 “Per” 结点的集合属性为 “0..1”，同时修改 Initial Lead Selection 属性为 “false”，如图 3-42 所示。

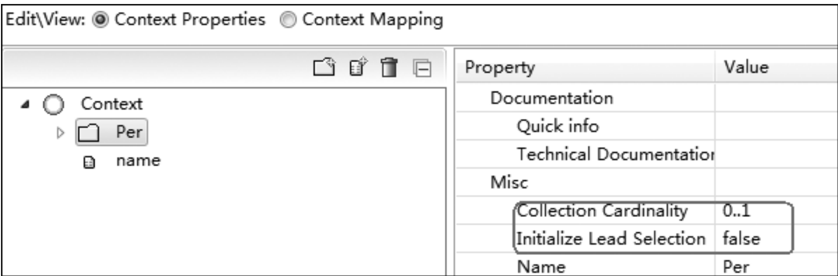


图 3-42　设置结点属性

2） 切换到 wdDoInit( )方法，保留以下代码（见图 3-43）：

```
wdComponentAPI. getMessageManager( ). reportSuccess ( " PerNode Size:" + wdContext. nodePer ( )
. size( ) );
IPerElement createAndAddPerElement = wdContext. nodePer( ). createAndAddPerElement( );
createAndAddPerElement. setId( "12" );
wdComponentAPI. getMessageManager( ). reportSuccess ( " PerNode Size:" + wdContext. nodePer ( )
. size( ) );
```

```
public void wdDoInit()
{
  //@@begin wdDoInit()

     // wdComponentAPI.getMessageManager().reportSuccess("Init");
     // wdContext.currentContextElement().setName("王朋");
     wdComponentAPI.getMessageManager().reportSuccess(
             "PerNode Size:" + wdContext.nodePer().size());
     IPerElement createAndAddPerElement = wdContext.nodePer()
             .createAndAddPerElement();
     createAndAddPerElement.setId("12");
     // createAndAddPerElement =
     // wdContext.nodePer().createAndAddPerElement();
     // createAndAddPerElement.setId("13");
     wdComponentAPI.getMessageManager().reportSuccess(
             "PerNode Size:" + wdContext.nodePer().size());
     // wdContext.createContextElement();

  //@@end
}
```

图 3-43　代码修改

3） 双击 “Button” 按钮，进入按钮事件的逻辑处理，并保留以下代码（见图 3-44）：

```
IWDMessageManager messageManager = wdComponentAPI. getMessageManager( );
messageManager. reportSuccess( "Id Value Method1:" + wdContext. nodePer( ). currentPerElement( ) );
```

```
public void onActionButton(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionButton(ServerEvent)
      IWDMessageManager messageManager = wdComponentAPI.getMessageManager();
      // messageManager.reportSuccess("It is wdThis:"
      // + wdThis.wdGetButtonAction().getText());
      // messageManager.reportSuccess("It is wdContext:"
      // + wdContext.currentContextElement());
      // messageManager.reportSuccess("It is ComponentAPI:"
      // + wdComponentAPI.getMessageManager());
      // messageManager.reportSuccess("It is ControllerAPI:"
      // + wdControllerAPI.getAction("Button").getText());
      // logger.errorT("It is Logger Error!:");
      // String name = wdContext.currentContextElement().getName();
      // messageManager.reportSuccess(name);
      messageManager.reportSuccess("Id Value Method1:"
              + wdContext.nodePer().currentPerElement());

      // messageManager
      // .reportSuccess("Id Value Method2:"
      // + ((IPerElement) (wdContext.nodePer().getElementAt(0)))
      // );

      // Modal Window
      // Confirmation Dialog
  //@@end
}
```

图 3-44　代码修改

4）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 3-45 所示。

5）单击“Button”按钮进行测试，运行效果如图 3-46 所示。

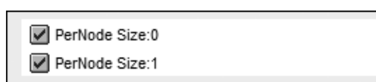


图 3-45　初始化运行效果



图 3-46　运行效果

可以看到打印的 Message 为 null，即 currentPerElement( ) 方法获取的是 Lead Selection 的条目，当 Lead Selection 为 false 时，该方法返回的是空指针。

（2）上下文属性实例 2

1）切换到 Context 视图，设置 node Per 的 Initialize Lead Selection 属性为“true”，如图 3-47 所示。

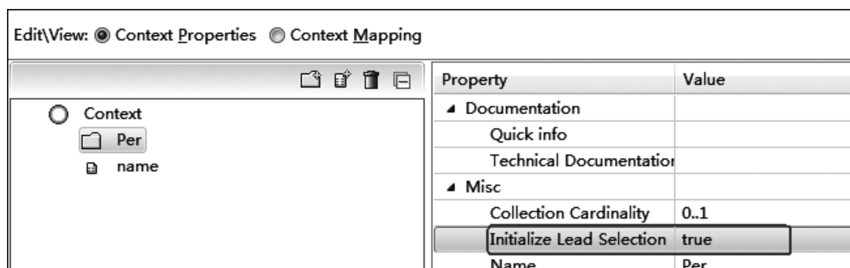


图 3-47　设置结点属性

2）保存所有修改，并重新编译、运行之后，即可进行相关测试，单击“Button”按钮，运行效果如图 3-48 所示。

从上面的结果可以看到，currentPerElement( ) 返回的值不为空，修改打印的代码可以打印出 currentElement 的 Id 属性值。



图 3-48　运行效果

（3）上下文属性实例 3

1）切换到 Context 视图，修改“Per”结点的集合属性为“0.. n”、Initialize Lead Selection 属性为“false”，保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 3-49 所示。

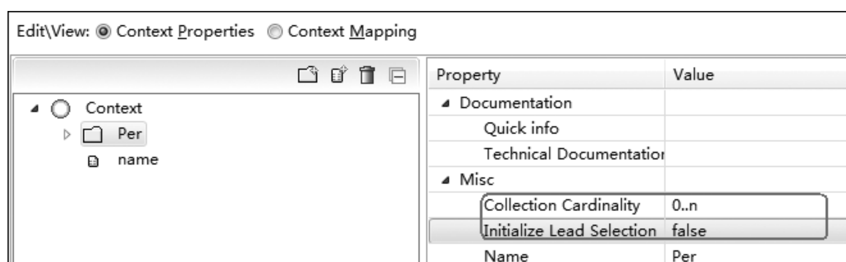


图 3-49　设置结点属性

2）单击“Button”按钮就能看到对应的提醒消息，current < NAME > Element( ) 返回的为空，如图 3-50 所示。



图 3-50　单击“Button”按钮后的运行效果

（4）上下文属性实例 4

1）再次修改 Initialize lead Selection 为“true”，保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 3-51 所示。

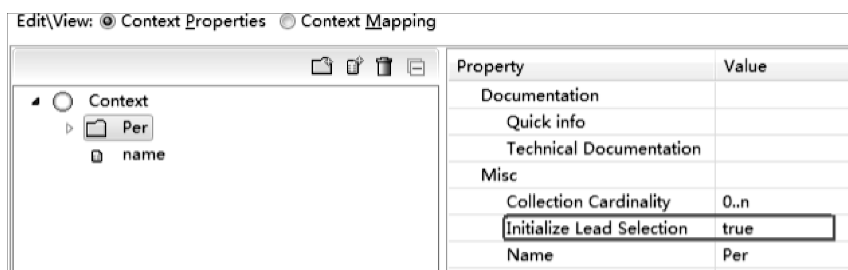


图 3-51　设置结点属性

2）单击“Button”按钮，可以看到 currentElement 返回的不为空，同时指向了对 node 第一条 Element 的引用，如图 3-52 所示。

3）切换到 OutLine 视图，右击“RootElement”，在弹出的快捷菜单中选择“Apply Template”，选项，如图 3-53 所示。

4）在弹出的对话框中选中“Table”类型，单击“Next”按钮，如图 3-54 所示。

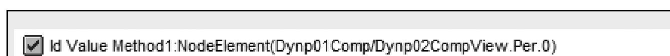


图 3-52　单击“Button”按钮后的运行效果

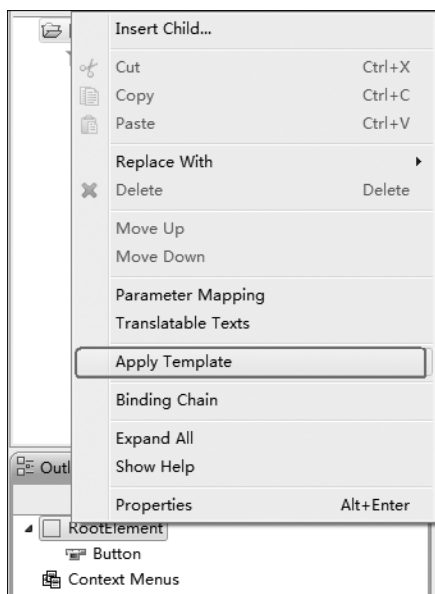


图 3-53　应用模板菜单选择

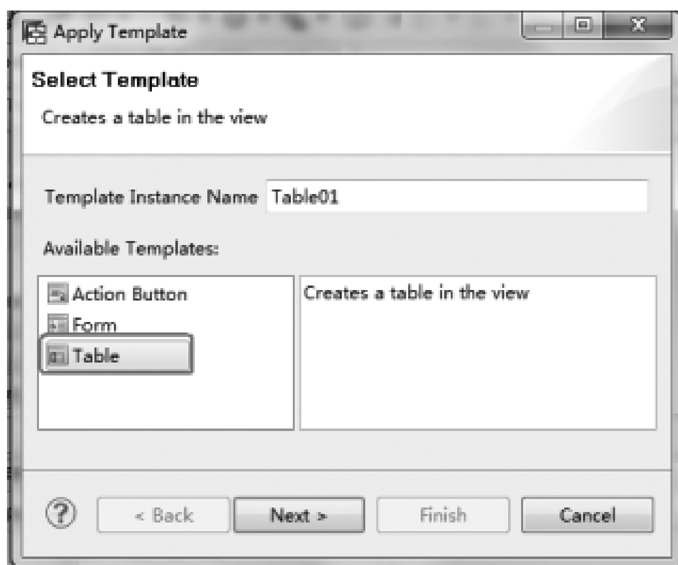


图 3-54　模板类型选择

5）选中“Per”结点和属性 Id，单击“Next”按钮完成创建，如图 3-55 所示。

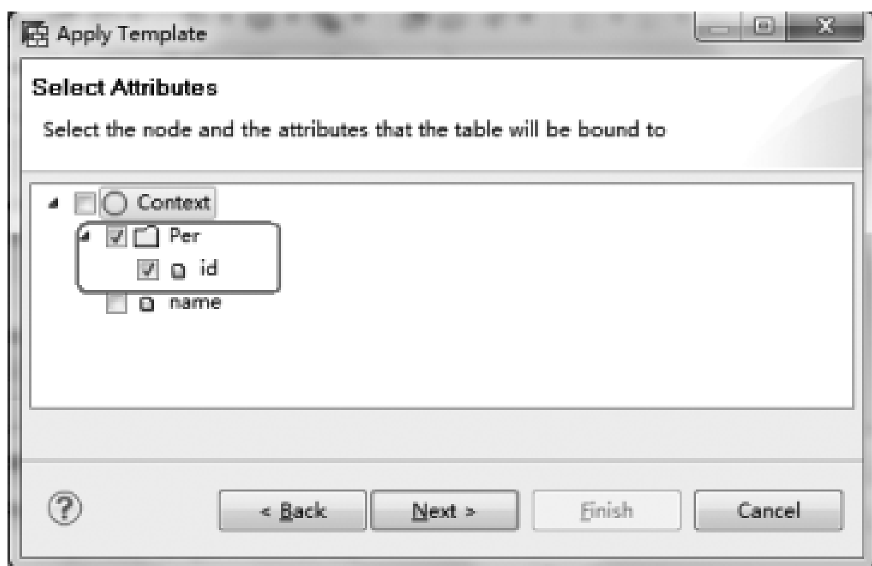


图 3-55　选择对应上下文

6）适当调整生成 Table_0 的宽度，如图 3-56 所示。

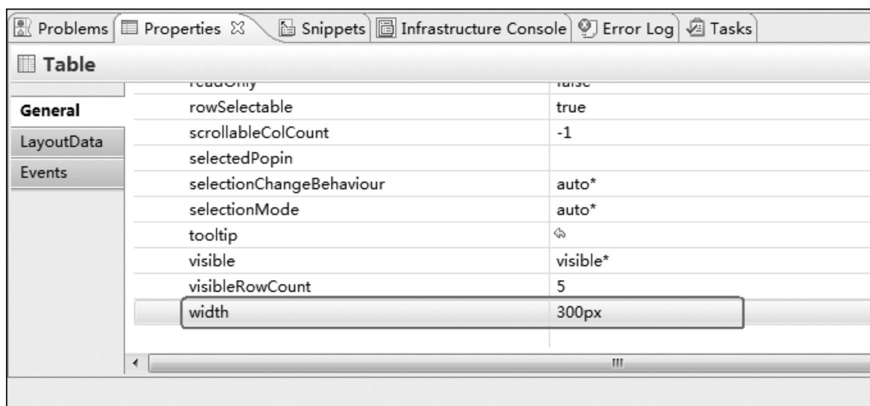


图 3-56　设置表格属性

7）切换到 wdDoInit( )方法，并添加以下代码（见图 3-57）：

```
createAndAddPerElement = wdContext. nodePer( ). createAndAddPerElement( ) ;
createAndAddPerElement. setId( "13" ) ;
```

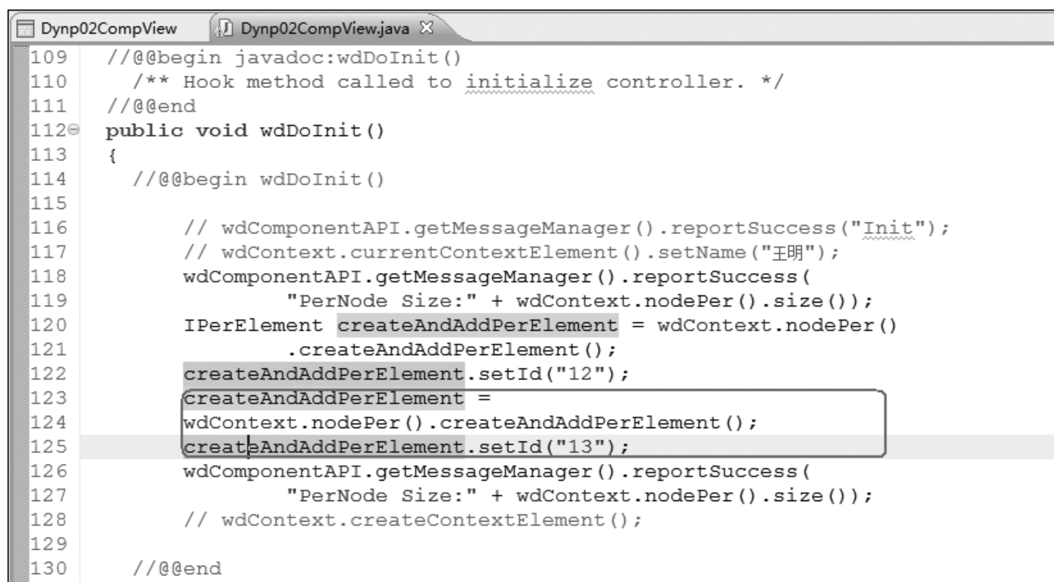


图 3-57　添加自定义代码

8）保存所有修改，并重新编译、运行之后，即可进行相关测试。初始化时，打印的 node Size 值为 2，Table_0 默认选中第一条，运行效果如图 3-58 所示。

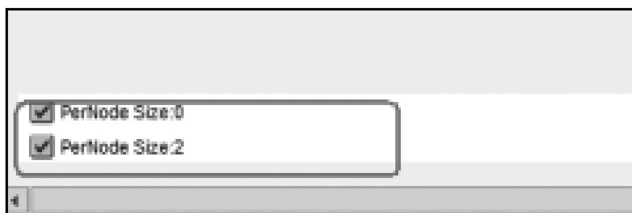


图 3-58　运行效果

9）单击“Button”按钮后，消息打印出一条 Element 的引用，如图 3-59 所示。



图 3-59　单击“Button”按钮后的运行效果

10）选中表格第二条行项目，并单击“Button”按钮，如图 3-60 所示。

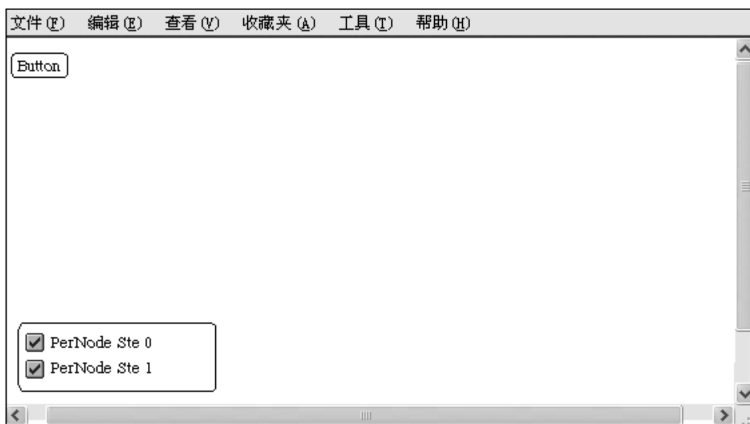


图 3-60　选择表格

11）单击“Button”按钮，运行效果如图 3-61 所示。

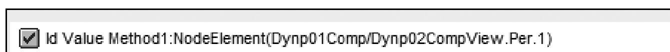


图 3-61　单击“Button”按钮后的运行效果

此时 current <NAME> Element 指向对第二条 Element 的引用，即当 Table_0 的 Selection Mode 为单选时（如果 Selection Mode 属性为 auto，则表格控件的选择模式由绑定的数据结构决定，上下文结点“Per”的选择属性为“0..1”，所以表格选择模式默认为单选），针对表格行项目的单击动作会对表格的首选项属性“Lead Selection”进行相应修改，随后 current <NAME> Element 返回修改后的首选项属性所指向的行项目的引用。

12）修改 node Per 的 selection cardinality 为 0..n、Initialize lead selection 为 false，同时切换到 Outline→Table_0→Properties→Selection Mode，并设置为 multiNoLead（multiNoLead 为多选模式，但选中行项目时不设置 Lead Selection），重新编译、部署，运行程序。从运行的结果可以看出，在上下文结点的“Initialize Lead Selection”为 false，同时以该结点为数据源的表格选择模式设置为 * No Lead 时，current <NAME> Element 返回始终为空，即结点并没有被赋值为 lead selected 的元素，运行效果如图 3-62 所示。

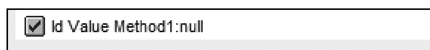


图 3-62　修改后的运行效果

3.2.4 视图的自定义方法

3.1.1 节介绍了视图中的一些标准方法，也可以在视图控制器（即视图类）中创建自定义方法（Methods）并定义方法的参数、方法的返回类型等内容。当需要创建一个自定义方法时，由于 Web Dynpro 的特殊性，开发人员并不能直接在视图控制器中编写代码，而是通过 NWDS 进行方法的创建。

1）首先需要在视图的“Properties”选项卡中找到“Methods”，在“Methods”下进行方法定义，如图 3-63 所示。

注：在 NetWeaver 7.0 及其之前版本的开发工具中，方法的定义是在与布局“Layout”并列的“Methods”中，然而在新版本的开发工具里上下文、方法、事件处理器等内容统一被包含在“Properties”选项卡中。

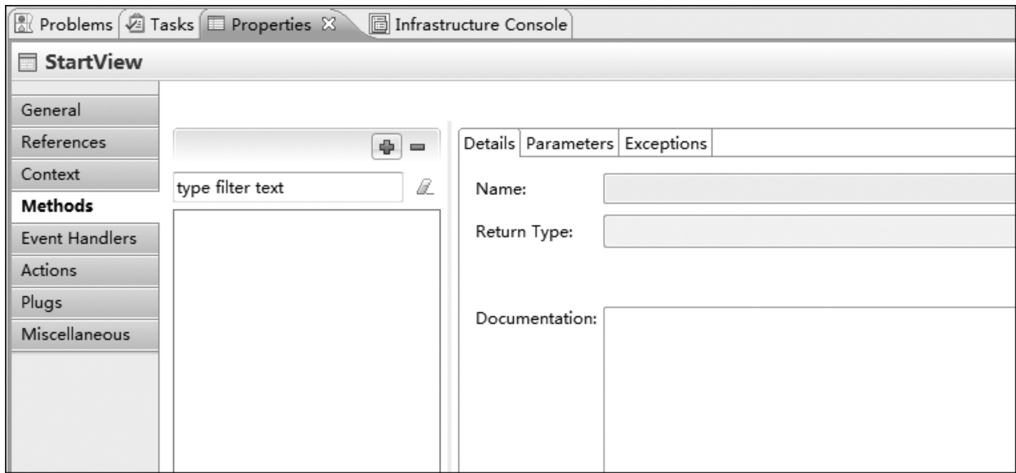


图 3-63 定义自定义方法

2）单击图 3-63 中的加号按钮，进行方法的创建，输入方法名并选择相应的返回类型，单击“Next”按钮，如图 3-64 所示。

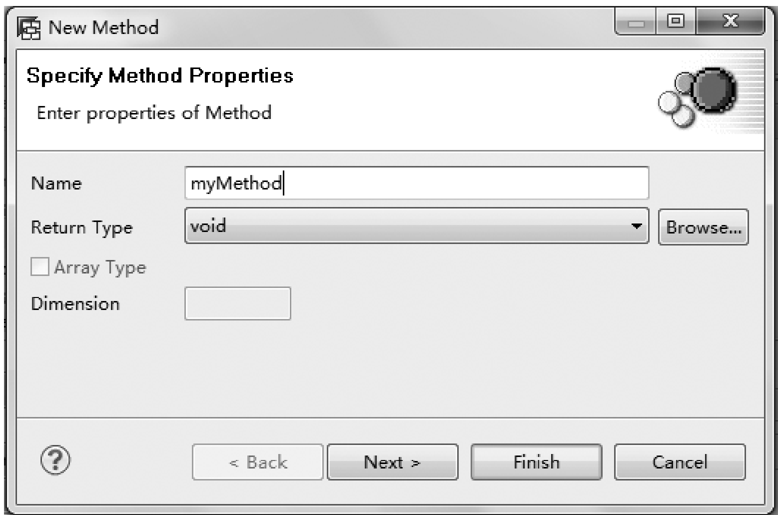


图 3-64 输入方法名称

3）如果方法存在参数，则单击“New”按钮进行定义，如图3-65所示。

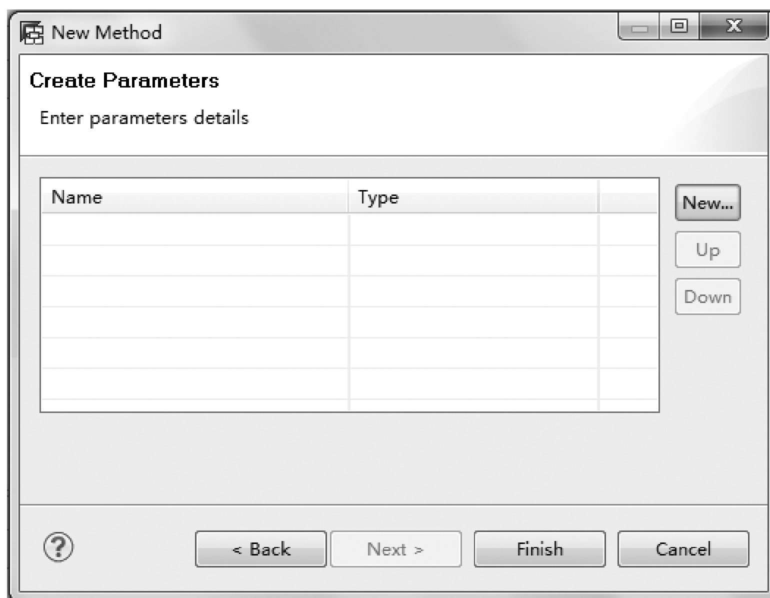


图3-65　定义方法的参数

4）在弹出的新建参数窗口中输入参数名称并选择对应的参数类型，单击“Finish”按钮，如图3-66所示。

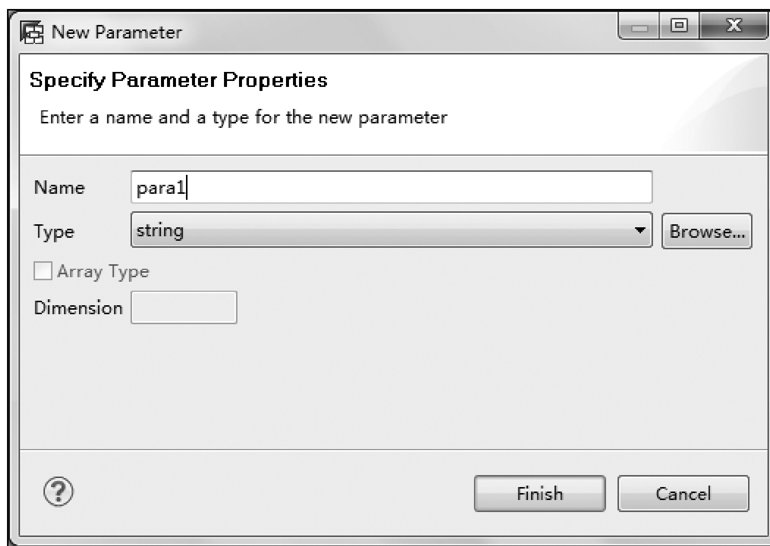


图3-66　定义新的参数

5）在完成参数创建后，单击“Finish”按钮，如图3-67所示。

6）至此完成了自定义方法的创建，在“Methods”下能看到对应的方法，如图3-68所示。

7）在保存之后（如果未保存，则双击后无法看到对应的方法），双击自定义方法就能进入控制器进行代码编写，如图3-69所示。

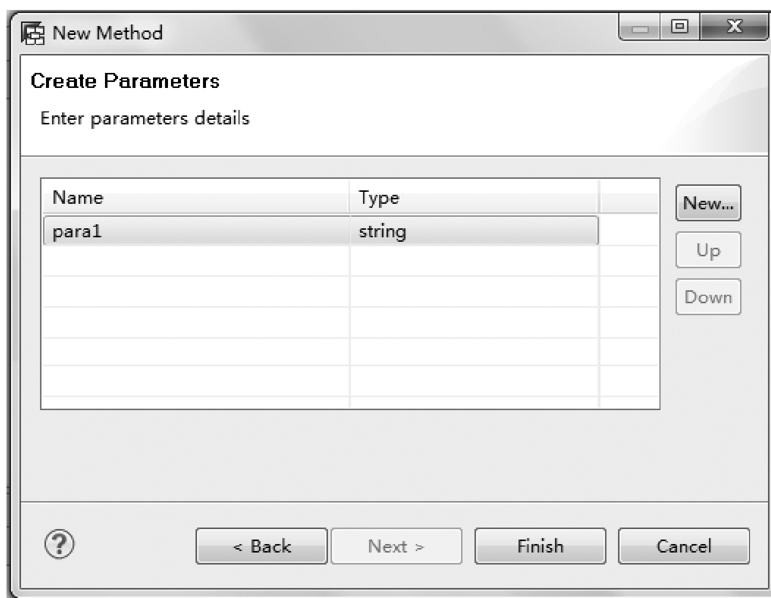


图 3-67　完成参数创建

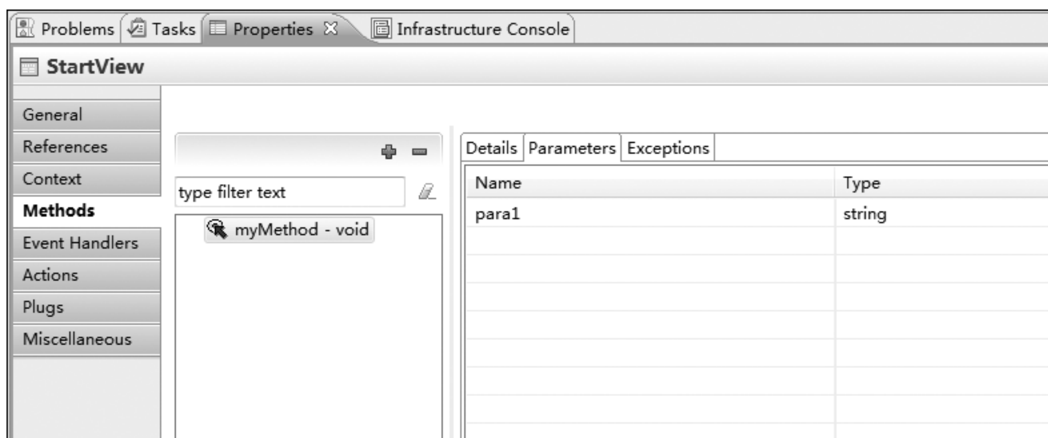
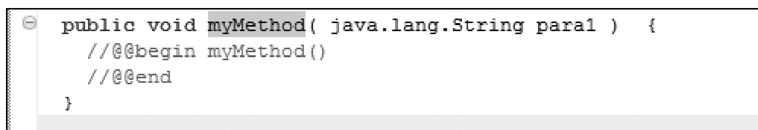


图 3-68　完成方法创建



```
public void myMethod( java.lang.String para1 )  {
  //@@begin myMethod()
  //@@end
}
```

图 3-69　编辑方法的代码

3.2.5　视图的事件处理器

事件处理器（Event handler）是 Web Dynpro 中一种特殊的方法，在 NetWeaver 7.0 及其之前的版本中，事件处理器与方法的创建是在同一选项卡中，通过两个单选项区分，而在新版本的开发工具中，事件处理器被单独放在一个“Event Handlers”中，如图 3-70 所示。

单击图 3-70 中的加号按钮，即可进行事件处理器的创建。在弹出的对话框中输入名称，选择需要订阅的事件，单击“Finish”按钮即可，如图 3-71 所示。

在 Web Dynpro 中事件（Event）来源于以下 3 种情况：

1）在组件控制器中创建的自定义事件（Event）。

2）视图页面的 UI 元素的特定动作产生的特定事件（onAction Event）。

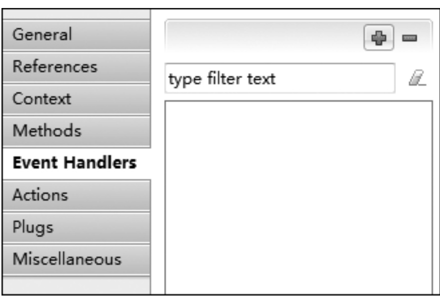


图 3-70　事件处理器选项卡

3）视图相互切换的导航插口对应的导航事件（Navigation Event）。

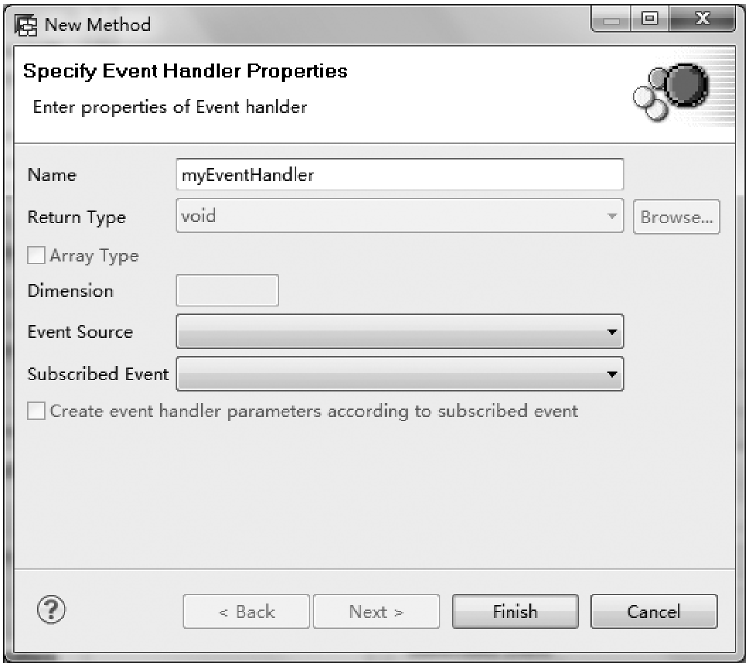


图 3-71　创建事件处理器

需要特别注意的是，一般情况下第一种自定义事件的情况需要开发人员手动地创建对应的事件处理器，且自定义事件只能在组件控制器中定义，视图、窗口中都无法创建自定义事件；对于后面两种情况，创建视图的动作或者插口后，开发工具都会自动生成对应的事件处理器，而动作只能在视图中定义。

1. 视图的动作

在实施项目中，大部分事件都来自于视图 UI 元素的动作，特定的 UI 元素会触发特定的事件，即产生了用户交互式活动，如按钮的单击、表格行的切换和下拉菜单的选择等。按钮事件的定义如图 3-72 所示。

选中视图中的 UI 元素，找到对应的“Events”子标签，就能通过单击“Create”创建动作，如图 3-73 所示。注：在 NetWeaver 7.0 及其之前版本中，Event 是在属性中进行定义，新版本已经分配了单独的标签进行配置。

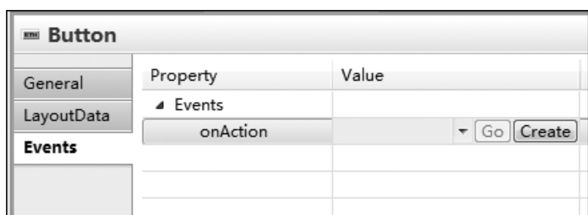


图 3-72　按钮事件的定义

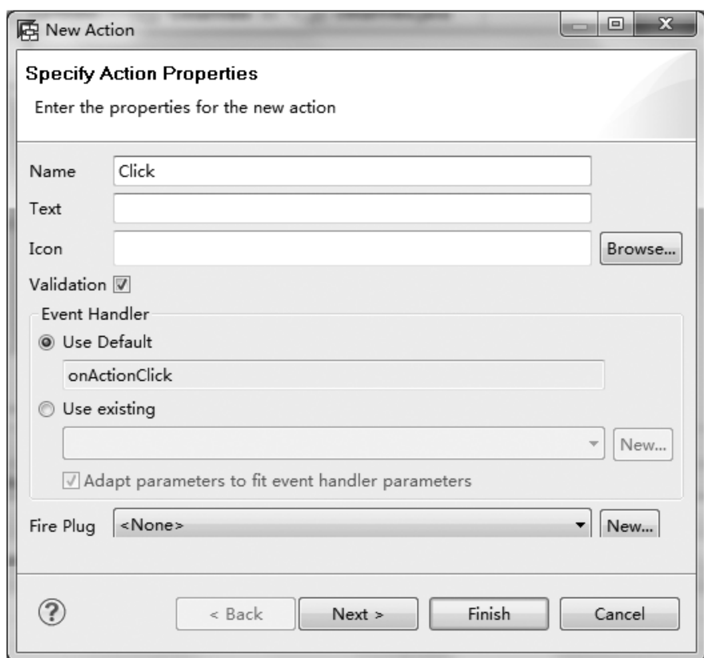


图 3-73　创建自定义动作

创建动作时需要填写的内容见表 3-3。

表 3-3　创建动作时需要填写的内容

标　　签	填 写 说 明
Name	动作的名称
Text	动作的描述文本
Icon	动作的图标（图标会对应出现在视图中）
Validation	是否校验
Event Handler（Use Default/Use existing）	选择（默认创建/已存在的）事件处理器
Fire Plug	选择需要触发的视图插口

如果选择默认创建事件处理器，保存之后会对应生成一个 onAction < NAME > 的事件处理器方法，这个方法就是用于处理客户端用户交互活动的事件。最终完成了从客户端的用户操作到视图页面元素，随后触发特定的事件，再通过事件处理器进行服务端逻辑处理的过程：用户（User）– 操作（Action）– 页面元素（UI Element）– 事件（Event）– 事件处理器

(Event Handler)。

2. 视图的插口

视图的插口是同一窗口中不同视图进行切换的出入口，在视图的“Properties”选项卡中可以找到对应的“Plugs”子标签，如图3-74所示。

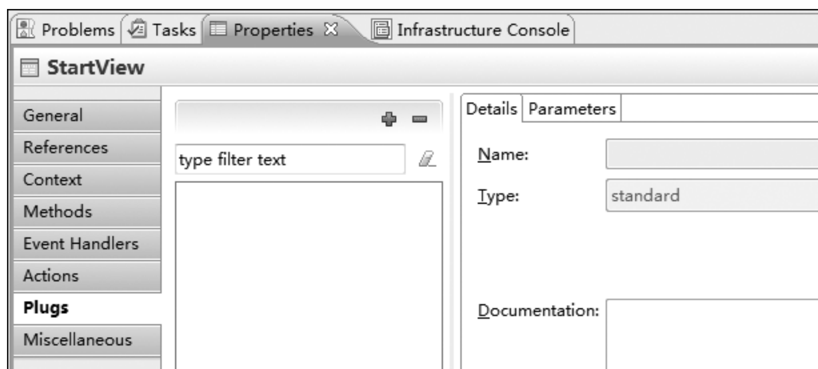


图3-74 “Plugs”子标签

它有Outbound Plug（红色）和Inbound Plug（蓝色）两种类型，分别作为出口和入口，如图3-75所示。



图3-75 两种不同类型

当Outbound被触发（fire）时，会产生一个导航事件（Navigation Event），根据在窗口中定义的导航链接，触发对应Inbound的事件处理器，同时视图页面进行切换。

开发实例：

1）在“Dynpro01Comp”组件中，右击“Views”视图，创建“Dynp01_PL01_View”和“Dynp01_PL02_View”两个视图，如图3-76所示。

2）右击“Windows”，创建一个窗口Dynp01CompPlugWin，如果使用已存在的窗口，请直接从第9）步开始进行，如图3-77所示。

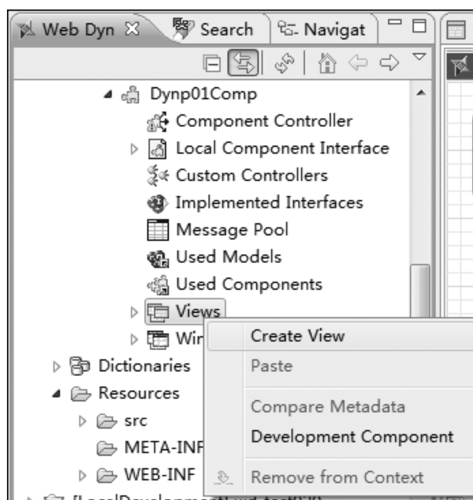


图3-76 创建视图

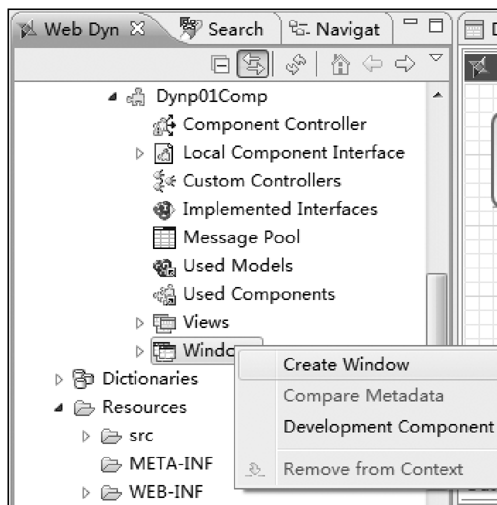


图3-77 创建新的窗口

3）创建窗口对应的接口视图，如图 3–78 所示。

4）输入接口视图的名称，单击“Finish”按钮，如图 3–79 所示。

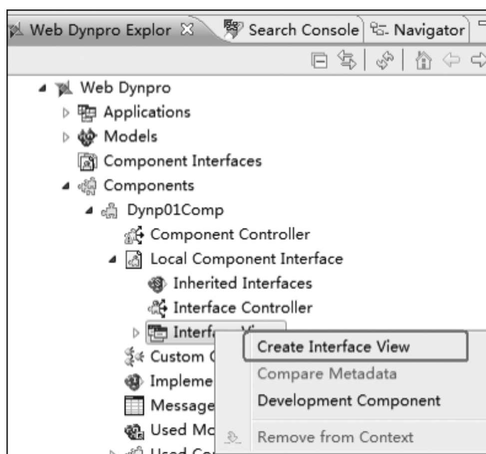


图 3–78　创建接口视图

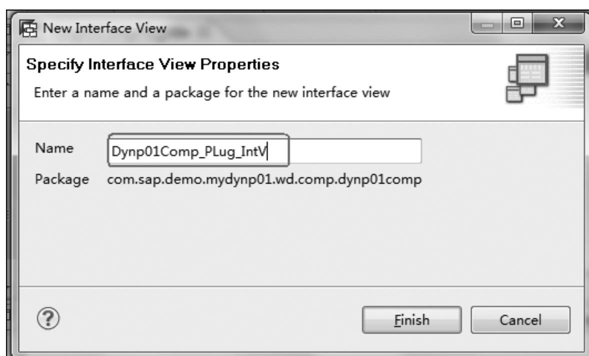


图 3–79　输入接口视图名称

5）由于接口视图对外公开，因此必须存在默认的 Inbound 插口。打开接口视图，单击左侧的“New”按钮，如图 3–80 所示。

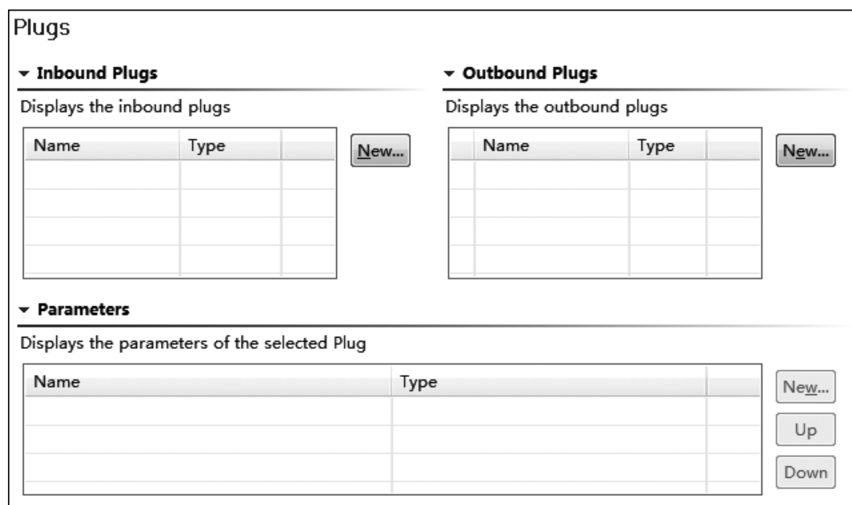


图 3–80　创建新插口

6）创建类型为“Startup”的插口，输入插口名称，如图 3–81 所示，单击“Finish”按钮并保存。

7）将接口视图与之前创建的窗口进行关联（如默认创建的窗口，将自动完成这些配置），接着打开 Dynp01CompPlugWin 窗口，在“Properties”选项卡中找到“InterfaceViews”子标签，单击创建视图按钮，如图 3–82 所示。

8）在弹出的对话框中选择刚才创建的接口视图“Dynp01Comp_Plug_IntV”（见图 3–83），单击“OK”按钮并保存。

9）所有的配置完成后，单击“保存”按钮保存所有修改内容，如图 3–84 所示。

10）打开窗口，在空白处右击，在弹出的快捷菜单中选择“Embed View”，如图 3–85 所示。

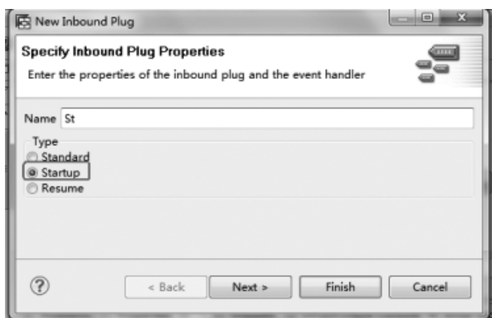


图 3-81　输入插口名称

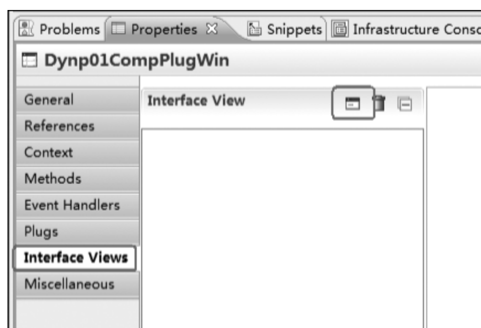


图 3-82　实现接口视图

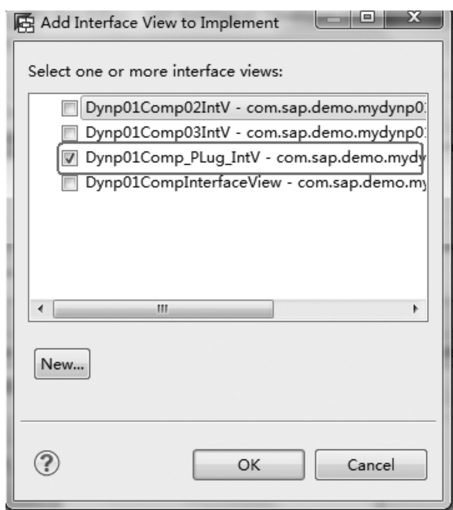


图 3-83　选择对应接口视图

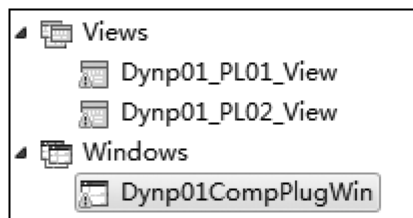


图 3-84　配置完成后的目录

11）在弹出的对话框中选中“Embed Existing View”单选按钮，嵌入已存在的视图，单击“Next”按钮，如图 3-86 所示。

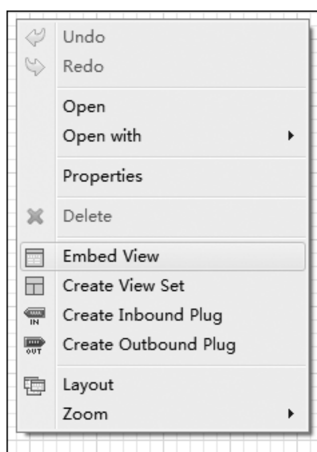


图 3-85　选择嵌入视图

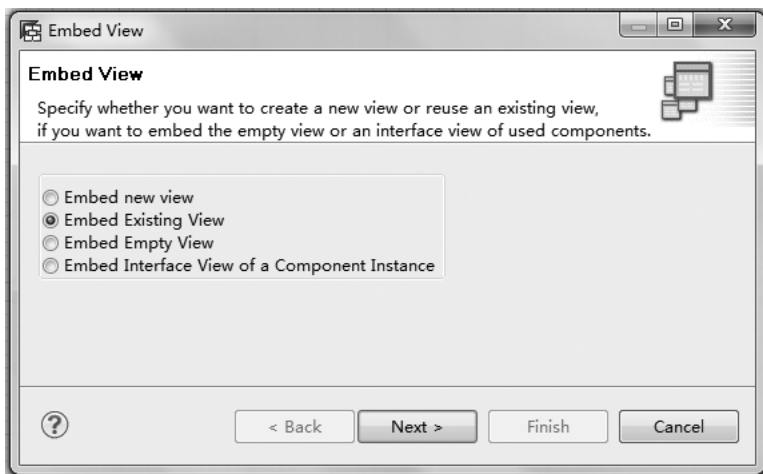


图 3-86　选择嵌入类型

12）选择对应的视图 Dynp01_PL01_View，单击“Finish”按钮，如图 3-87 所示。

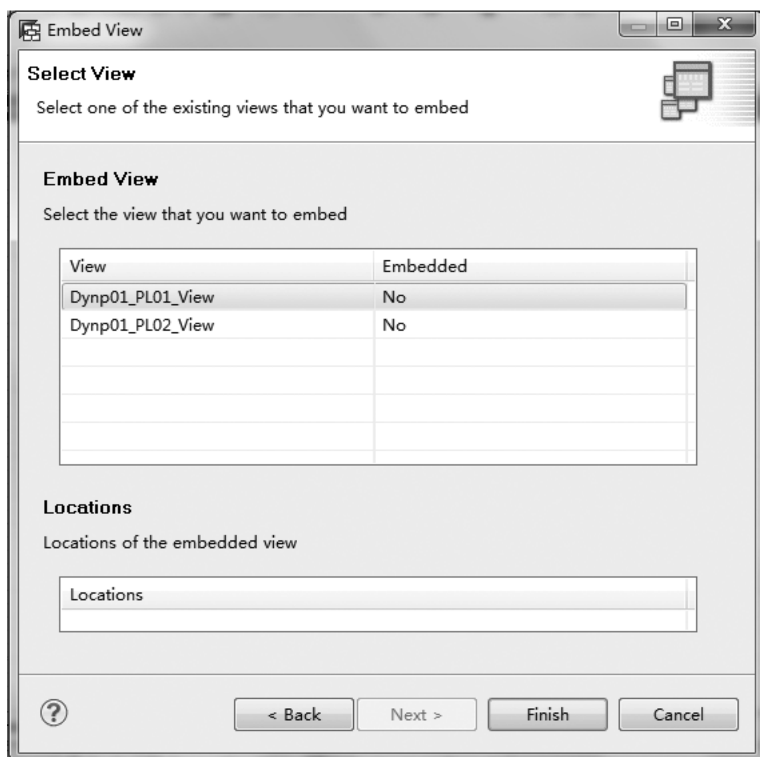


图 3-87　选择嵌入视图

13）重复步骤 4）~ 步骤 6），将另一个视图“Dynp01_PL02_View”嵌入到窗口中，如图 3-88 所示。

14）将鼠标移至“Dynp01_PL01_View”上，通过弹出的快捷菜单分别创建“Inbound”插口与“OutBound”插口，如图 3-89 所示。

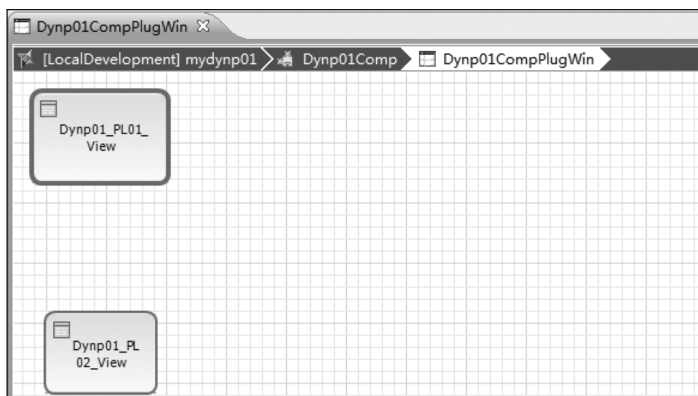


图 3-88　完成嵌入后的效果

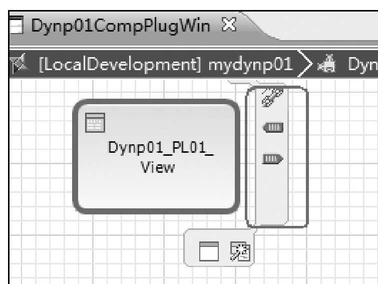


图 3-89　通过快捷菜单创建插口

15）单击对应的按钮，即可自动创建对应的插口。图 3-90 所示为创建的“Inbound”插口。

16）图 3-91 所示为创建“Outbound”插口。

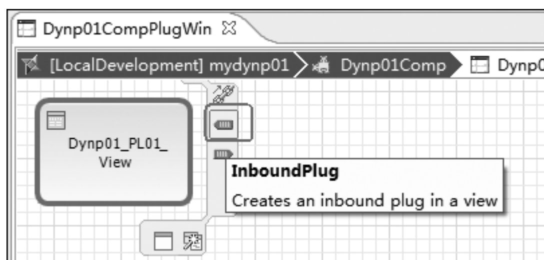


图 3-90　创建“Inbound”插口按钮

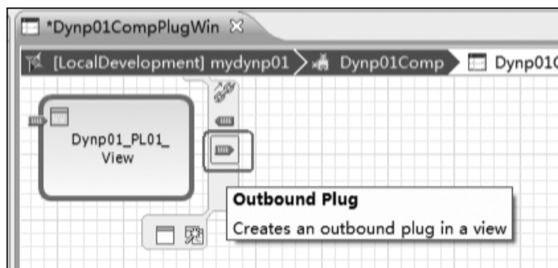


图 3-91　创建“Outbound”插口按钮

17）同样地，为 Dynp01_PL02_View 视图创建两个插口后，单击右侧的“Link”按钮，创建导航链接，如图 3-92 所示。

注：只能将 Outbound Plug 与 Inbound Plug 相连，同类型的插口无法创建导航链接。

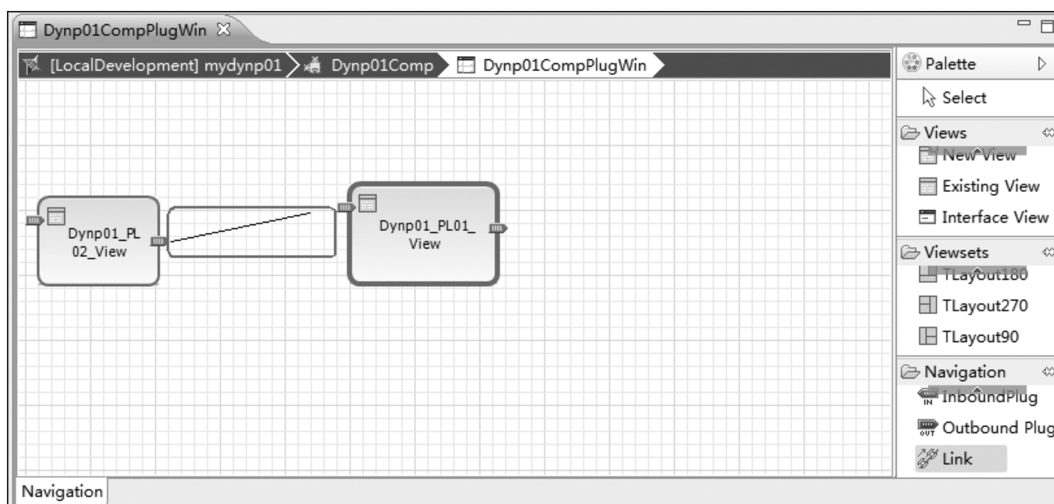


图 3-92　创建导航链接

18）在弹出的窗口中选勾选第二项：由向导创建默认按钮以执行跳转动作，如图 3-93 所示。如果已手动创建按钮，则不需要勾选可跳过。

19）用同样的方法创建一个返回的导航链接，即完成关联，如图 3-94 所示。

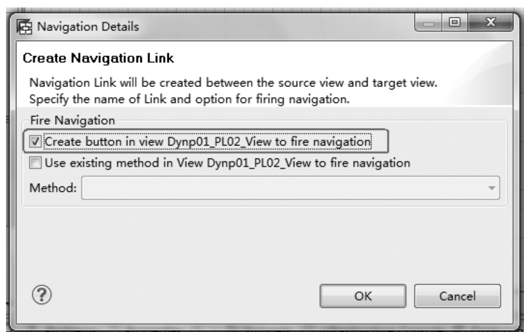


图 3-93　自动创建按钮选项

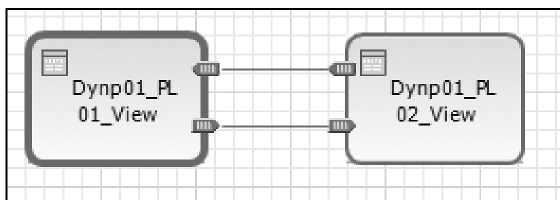


图 3-94　创建返回导航

20）右击“Applications”创建一个应用，如图 3-95 所示。

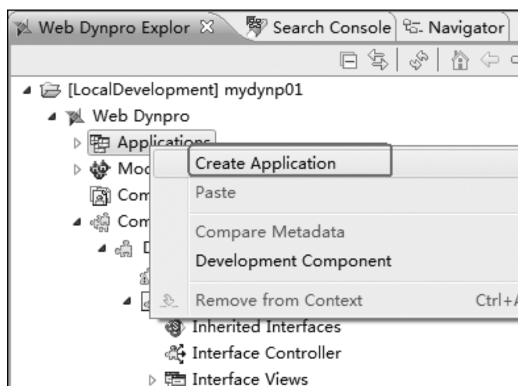


图 3-95　创建应用程序

21）自定义应用名称和包名，单击“Next”按钮，如图 3-96 所示。

22）选择应用运行时需要展示的接口视图，如图 3-97 所示。

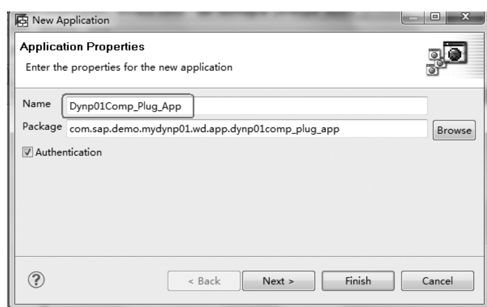


图 3-96　输入应用名称

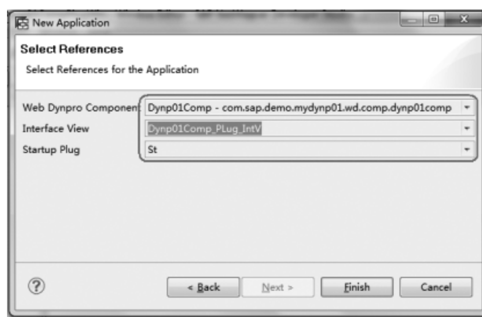


图 3-97　选择对应的接口视图

23）保存所有修改，并重新编译、运行之后，即可进行“Dynp01Comp_Plug_App”应用测试，运行效果如图 3-98 所示。

24）单击“ToDynp01_PL01_View”按钮即可跳转到 PL01_View，再次单击该按钮，则跳回，如图 3-99 所示。

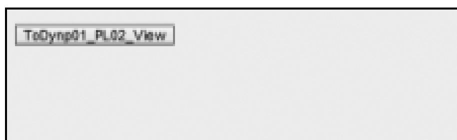


图 3-98　运行效果

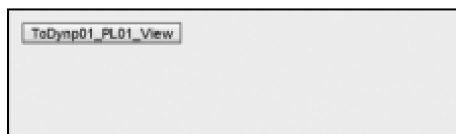


图 3-99　单击按钮运行效果

注：自动生成代码是 NWDS 一个强大的功能。在本实例中，按钮的定义、导航事件的处理和触发都是在创建导航链接时。

3.2.6　视图的 UI 元素

1. UI 元素介绍

Web Dynpro 将 Java 中的图形化开发进行了彻底的封装，在 NWDS 的视图开发中可以直

接在视图编辑器中通过图形化的开发方式进行界面元素的修改和调整，大大减小了代码编写的工作量，同时也降低了开发人员的学习难度。接触过网页开发或其他页面搭建的用户，应该经常会接触到“控件”或者“UI 控件”这样的描述，而在 Web Dynpro 中确切的叫法是“用户界面元素”或“UI 元素”（User Interface Element）。

Web Dynpro 中的 UI 元素有许多不同的类型，包括简单元素（如文本框、标签、按钮等元素）、复杂元素（如表格、列表、工具栏等）以及特殊元素（如交互表单、定时器、图片等），方便开发人员满足不同的业务需求，如图 3-100 所示。

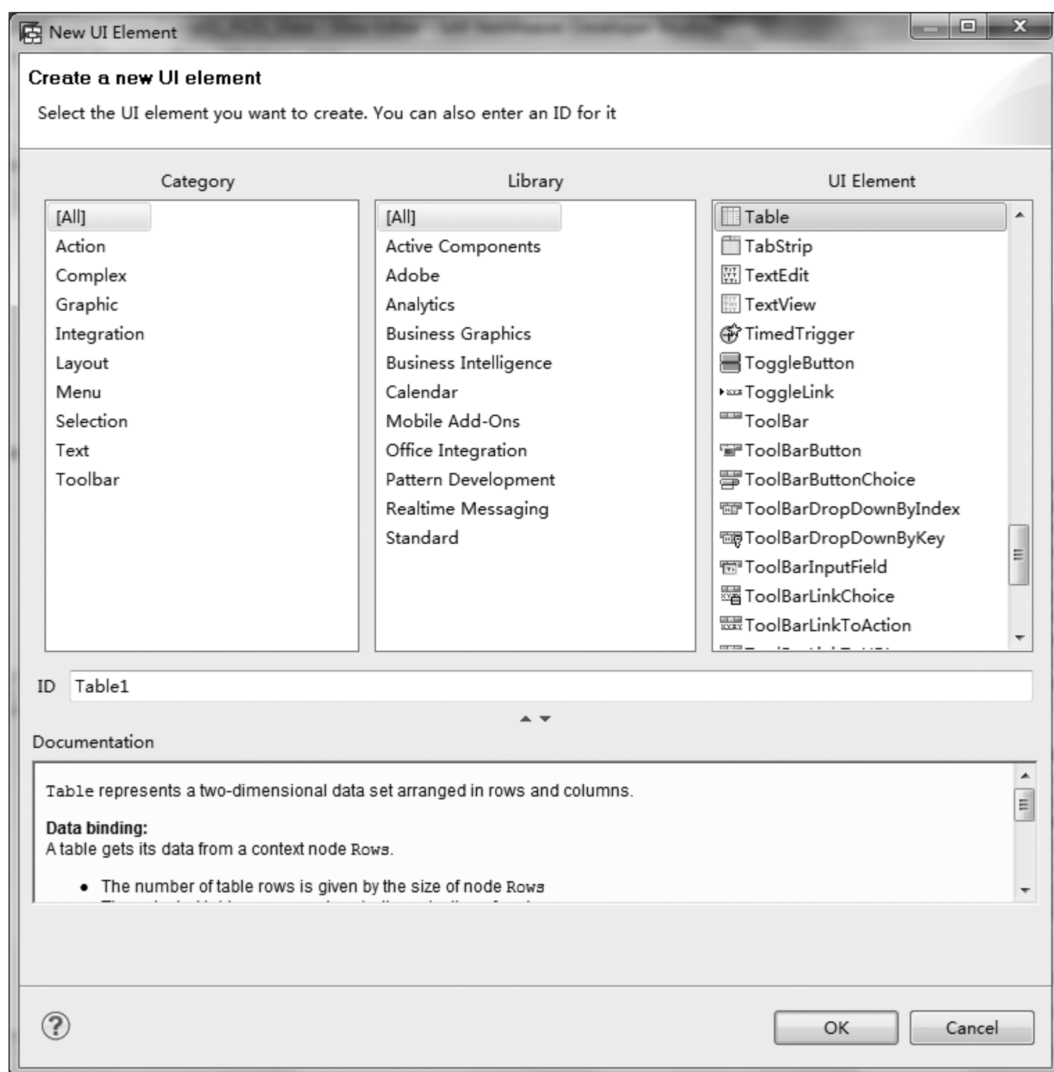


图 3-100　UI 元素的各种类型

2. 动态创建视图页面元素

通常情况下，在设计视图的布局（Layout）和 UI 元素（UI Element）时，都是在数据定义或数据结构已知的情况下完成页面元素的创建、属性的设定以及页面元素的整体布局，在数据结构本身并不确定的前提下，就需要能够在程序运行时根据实时运行结果得到的数据结

构动态地进行页面控件的生成了。

当然对于部分静态页面的需求或者一些通用布局的页面，也可通过页面元素的 API 进行创建，但一定要注意程序的运行效率，切勿为了写代码而写代码，从而变得舍本逐末。

事实上，Web Dynpro 不仅支持动态的页面元素创建，还支持动态上下文（Context）的创建和动作（Action）的动态分配，这里仅以动态的页面元素创建为例进行讲解，后两者在实际项目中遇到的情况并不多。

在 Web Dynpro 中，视图控制器通过“wdModifyView”方法提供给开发人员动态创建视图页面元素的唯一入口。该方法有两个参数，分别是类型为“IWDView”的视图实例与类型为布尔型的“firstTime”。其中，视图实例可以用于获取、创建当前页面的实例以及其中所有 UI 元素，并可通过标准 API 对元素属性进行动态地设置和调整；“firstTime”参数用于标识方法是否为第一次执行，通过“firstTime”可以控制方法逻辑的执行频率。如“firstTime == true”时，表示现在是程序页面第一次初始化，可写入初始化的动态页面元素创建的代码。下面通过一个开发实例来看一下动态创建在项目实施中的实际应用。

开发实例：

目标：单击页面上的按钮，动态创建两个标签和两个文本框。

1）在 Web Dynpro 项目的视图中添加一个按钮，如图 3-101 所示。

2）上下文中新建一个布尔型特性“isCreateForm”，如图 3-102 所示。



图 3-101　添加按钮元素



图 3-102　创建上下文特性

3）编写触发按钮的动作，当单击“触发”按钮时，动作对应的逻辑代码就会修改变量，并在运行时新建上下文特性。

```
//@@begin onActionCreateForm(ServerEvent)
    //设置变量
    wdContext.currentContextElement().setIsCreateForm(true);
    //创建 Context Node
    IWDNodeInfo perInfo = wdContext.wdGetAPI().getRootNodeInfo().addChild(
        "PerInfo", null, true, CMICardinality.ONE,
        CMICardinality.ZERO_TO_ONE, true, null);
    //Node 下新增两个属性:Id 与 Name
    //新建属性
    IWDAttributeInfo id = perInfo.addAttribute("Id",
        "ddic:com.sap.dictionary.string");
    IWDNodeElement currentElement = wdContext.getChildNode("PerInfo", 0)
        .getCurrentElement();
    //设置属性值
    currentElement.setAttributeValue("Id", "P001");
    //新建属性
    IWDAttributeInfo name = perInfo.addAttribute("Name",
        "ddic:com.sap.dictionary.string");
```

```
        //设置属性值
        currentElement.setAttributeValue("Name","Blus");
    //@@end
```

4）在“wdModifyView”方法中通过控制变量，生成 UI 元素，方法在动作执行完毕后执行。

```
    //@@begin wdDoModifyView
        //获取控制变量
        boolean isCreateForm = wdContext.currentContextElement()
                .getIsCreateForm();
        if(isCreateForm){
            //获取 view 根结点
            IWDTransparentContainer container = (IWDTransparentContainer)view
                    .getElement("RootElement");
            //判断是否已创建 TransparentContainer
            IWDTransparentContainer IWDTransparentContainer01 = null;
            if(view.getElement("IWDTransparentContainer01") == null){
                //创建 TransparentContainer
                IWDTransparentContainer01 = view.createElement(
                        IWDTransparentContainer.class,
                        "IWDTransparentContainer01");
                //是否属于 LayoutContainer
                IWDTransparentContainer01.setIsLayoutContainer(true);
                //设置 GridLayout
                IWDGridLayout gridLayout = IWDTransparentContainer01
                        .createLayout(IWDGridLayout.class);
                //GridLayout 布局
                gridLayout.setColCount(2);
                //设置 TransparentContainer 可见性
                IWDTransparentContainer01.setVisible(WDVisibility.VISIBLE);
                //新建的 Container 添加至根结点
                container.addChild(IWDTransparentContainer01);
            }
            //获取运行时创建的 Node,该处可以设置为动态变量
            IWDNode childNode = wdContext.getChildNode("PerInfo",0);
            //获取新建的 currentNodeElement
            IWDNodeElement currentElement = childNode.getCurrentElement();
            //通过 NodeInfo 获取所有属性
            List<? extends IWDAttributeInfo> attributes = childNode
                    .getNodeInfo().getAttributes();
            //遍历所有属性并动态创建 UI 元素,该处每一个属性生成对应的 Label 与 InputField
            for(IWDAttributeInfo attrinfo : attributes){
                //获取属性名称
                String attr = attrinfo.getName();
                //根据属性名称获取属性值
                String attrValue = (String)currentElement
                        .getAttributeValue(attr);
                //判断是否已有对应的控件
                if(view.getElement("Label" + attr) == null){
                    //创建 Node 属性对应的 label 元素
                    IWDLabel labelId = (IWDLabel)view.createElement(
                    IWDLabel.class,"Label" + attr);
                    //设置元素可见性
                    labelId.setVisible(WDVisibility.VISIBLE);
                    //设置 labeltext
```

```
                labelId. setText( attr) ;
                //添加 label 元素至 TransparentContainer
                IWDTransparentContainer01. addChild( labelId) ;
            }
            //判断是否已有对应的控件
            if( view. getElement( "InputField" + attr) == null) {
                //创建 Node 属性对应的 InputField 元素
                IWDInputField inputfield = ( IWDInputField) view
                . createElement( IWDInputField. class, "InputField"
                 + attr) ;
                //控件绑定属性值,该值为 path
                inputfield. bindValue( "PerInfo. " + attr) ;
                //设置元素可见性
                inputfield. setVisible( WDVisibility. VISIBLE) ;
                //添加 InputField 元素至 TransparentContainer
                IWDTransparentContainer01. addChild( inputfield) ;
            }
        }
    }
//@ @ end
```

5）保存所有修改，重新编译、运行之后，即可进行相关测试，运行效果如图 3–103 所示。

6）单击“创建表单”按钮，页面动态创建的 UI 元素如图 3–104 所示。



图 3–103　运行效果

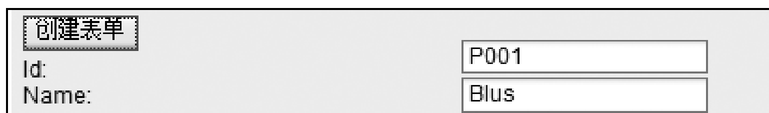


图 3–104　单击“创建表单”按钮的运行效果

上面的例子是一个简单的根据上下文动态生成界面元素的过程，在运行时新增了两个上下文特性，然后根据这些特性的名称、类型等动态地创建了视图中的 UI 元素，即每个属性对应一组文本标签和输入框元素。

实际上，页面的所有元素都有其 IWDView 的 UI Element API，通过这些 API 可以创建新的元素，也可以编辑这些元素的属性，所有的元素再通过上下级的关联来构造出整个树形结构的页面布局。

动态的页面元素解决了数据结构不确定的情况下页面构造的问题，从整个代码 API 构造的过程可以看出，WEBDYNPRO 提供的模板布局的快速。由于 wdDoModifyView 方法的执行频率很高，因此不建议在该方法中编写更多的逻辑（**注：**尤其不允许编写数据相关逻辑，因为这与 MVC 架构不符，且会影响后期代码维护）。需要特别注意，wdDoModifyView 频繁执行时客户端的性能及服务端的响应时间。

3.3　窗口

本节将具体介绍 Web Dynpro 中的窗口。

3.3.1　窗口的属性

窗口是视图的载体，每个窗口都可以嵌入一个或多个视图或者视图集，而视图集中又可

以嵌入一个或多个视图或者视图集，就像是一个不断生长的树状结构（在老版本的开发工具，可以在工程的目录中将窗口展开看到类似的结构但在新版本中无法看到这一结构）。但要注意的是，在每一块显示区域中只能有一个视图设置为“默认可见”，不允许有两个或两个以上视图均为“默认可见”的状态。

当在 Web Dynpro 中创建一个窗口时，并不仅仅创建了一个窗口对象，同时也创建了以下内容：

1）所有视图在组件中的可视化接口。

2）多个视图在同一页面中的布局和位置。

3）多个视图是否可见，以及相互之间如何切换。

需要注意的是，窗口实现了对应的接口视图，即与接口视图（Interface View）对应的是窗口，而不是视图。

进入 NWDS 开发者工作室，以 3.1.5 中创建的 Dynp01CompPlugWin 窗口为例，在“Properties”中包含了若干子标签，如图 3-105 所示。

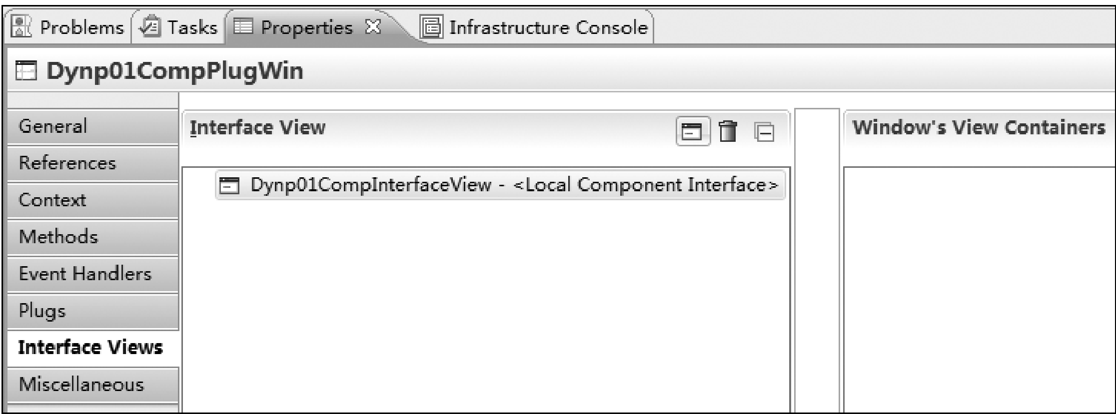


图 3-105　窗口的属性

属性子标签及功能说明见表 3-4。

表 3-4　属性子标签及功能说明

属性子标签	功 能 说 明
General	基本信息
References	引用，嵌入其他组件视图时的控制器列表
Context	上下文，同视图
Methods	方法，同视图
Event Handlers	事件处理器，同视图
Plugs	插口，同视图
Interface Views	接口视图，指定当前窗口实现的具体接口视图
Miscellaneous	其他内容

这些窗口中的属性子标签与视图的属性用法类似，如上下文、方法、事件处理器和插口等，这里不再赘述。

3.3.2 窗口与接口视图

在 Web Dynpro 的项目架构中，应用程序（Application）作为整个应用的入口点，窗口（Window）为应用的界面，接口视图（InterfaceView）则是 Web Dynpro 程序视图的入口点，任何新建的接口视图都需要一个默认的窗口实例去实现，而接口视图也衔接了入口点与界面的关联。

开发实例：

1）展开上一节中创建的 Web Dynpro 项目，右击“Interface Views”，在弹出的快捷菜单中选择“Create Interface View”创建一个接口视图。在弹出的对话框中输入接口视图的名称，单击“Finish”按钮完成创建，如图 3-106 所示。

2）在创建完成后，双击打开对应的接口视图，单击“New”按钮，创建一个“Inbound”的插口，如图 3-107 所示。

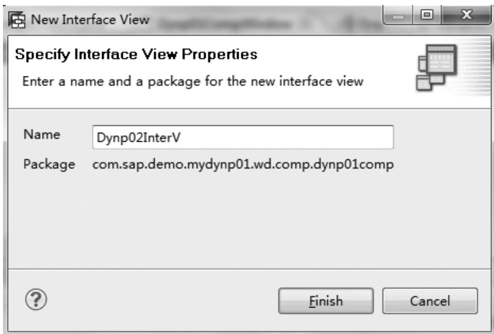


图 3-106　输入接口视图名称

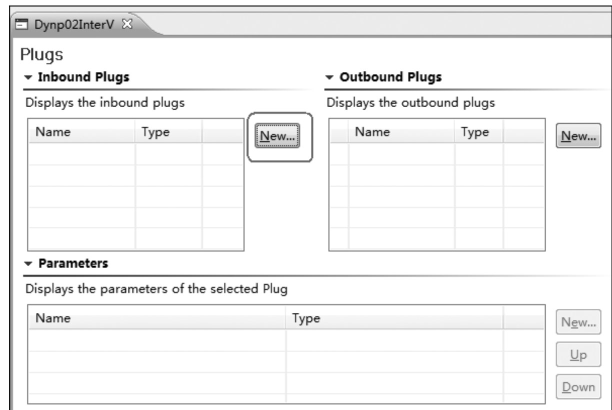


图 3-107　创建插口

3）在弹出的对话框中选中“Startup”单选按钮启动类型的查看，并输入自定义名称“St”，单击“Finish”按钮完成插口的创建，如图 3-108 所示。

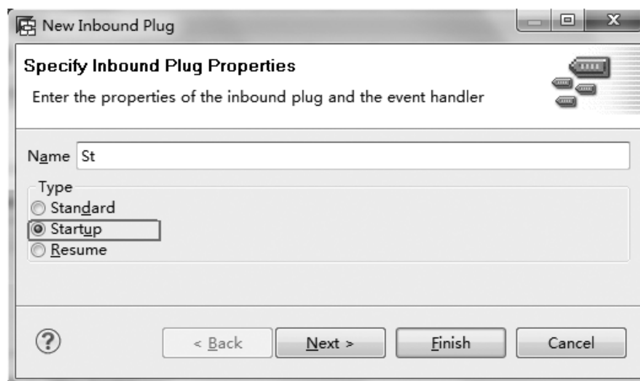


图 3-108　选择插口类型

4）保存所有修改后，会在编译时报错（见图 3-109）。在日志中查看具体的错误信息，由于刚创建的接口视图还未被窗口实现，因此编译无法通过。

5）下面创建一个窗口，并实现新建的接口视图。首先右击“Windows”，在弹出的快捷菜单中选择“Create Window”创建一个窗口，在弹出的对话框中输入自定义窗口名称，如

图 3-110 所示。

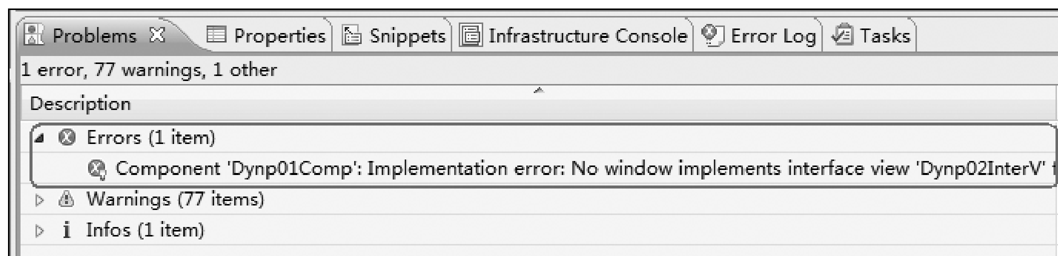


图 3-109　报错消息

6）单击“Add”按钮添加对应实现的接口视图，在弹出的窗口中选择刚才新建的接口视图“Dynp02InterV”，单击“OK”按钮，如图 3-111 所示。

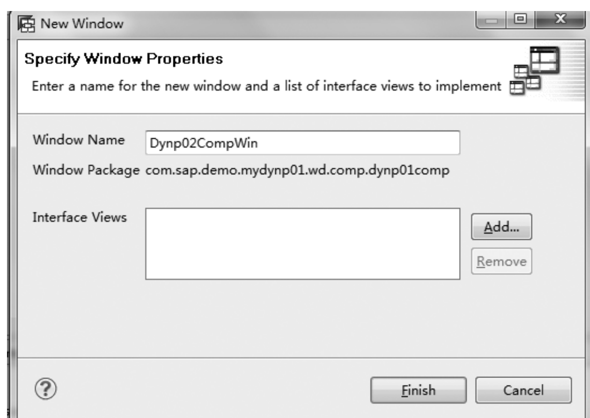


图 3-110　输入窗口名称

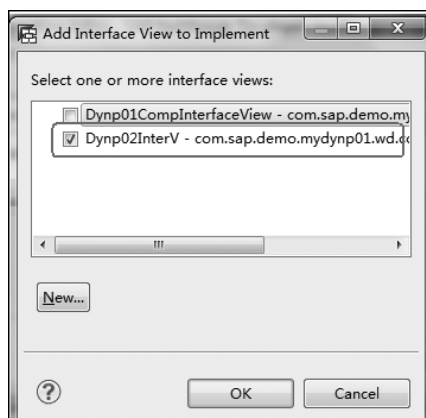


图 3-111　选择对应视图

7）添加接口视图后（见图 3-112），单击“Finish”按钮完成接口视图与窗口的手动创建。

8）在手动创建完接口视图后，再实际应用一下，看一下其他组件怎么引用接口视图。首先在项目中新建一个组件“Dynp02Comp”，在创建时选中“Default Window and Views”复选框，单击“Finish”按钮，如图 3-113 所示。

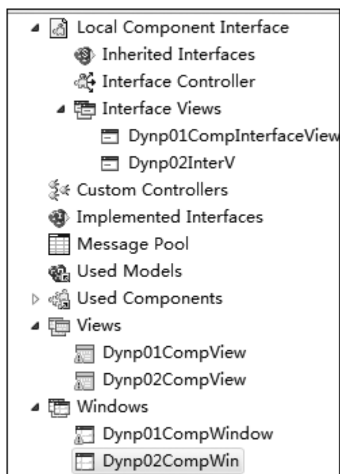


图 3-112　创建后的项目结构

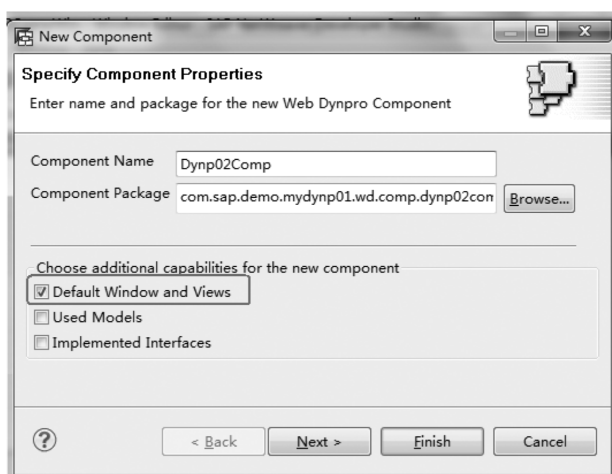


图 3-113　创建新的组件

9）创建完成新组件后，就能在 Web Dynpro 项目目录中看到具体内容，如图 3-114 所示。

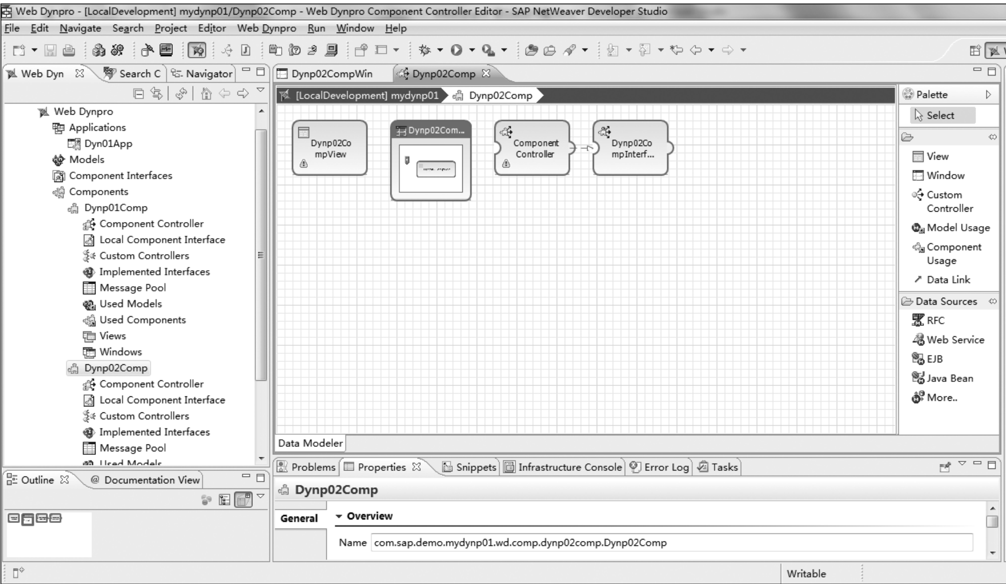


图 3-114　创建后的项目结构

10）双击打开手动创建的“Dynp02CompWin”窗口，在右侧空白处右击，在弹出的快捷菜单中选择“Embed View”选项，在窗口中嵌入视图，如图 3-115 所示。

11）在弹出的对话框中选择嵌入一个接口视图，如图 3-116 所示。

12）由于当前组件还未添加引用组件，因此先单击“Add”按钮，如图 3-117 所示。

13）在弹出的对话框中浏览并选中后来新建的“Dynp02Comp”组件（见图 3-118），单击“OK”按钮。

14）输入引用组件的名称，单击“Finish”按钮，如图 3-119 所示。

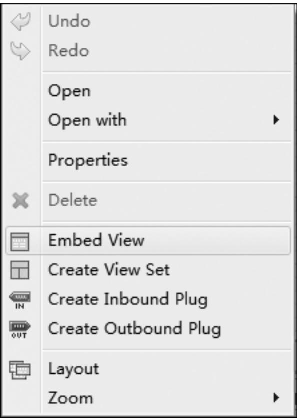


图 3-115　嵌入视图

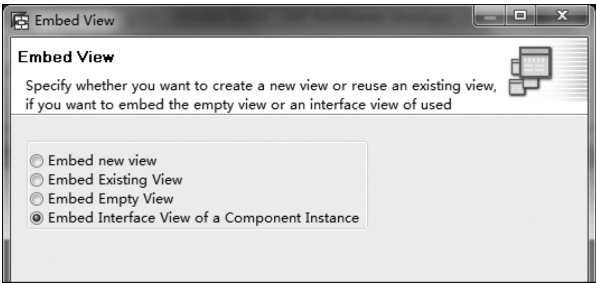


图 3-116　选择嵌入视图类型

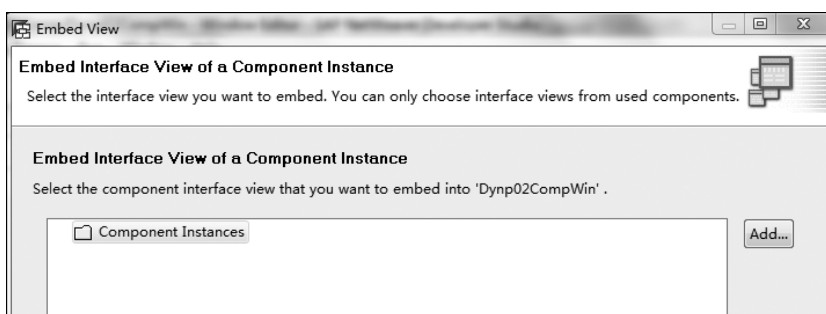


图 3-117　选择对应组件

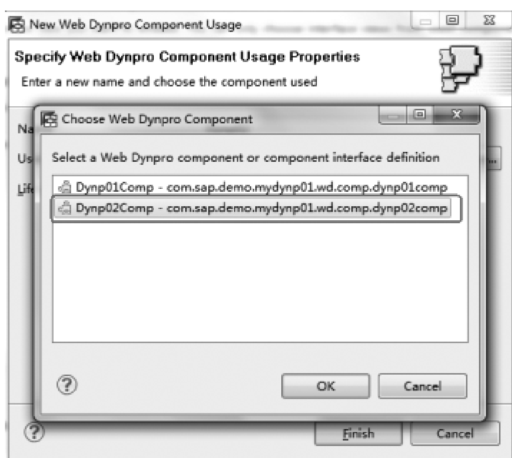


图 3-118　添加对应组件

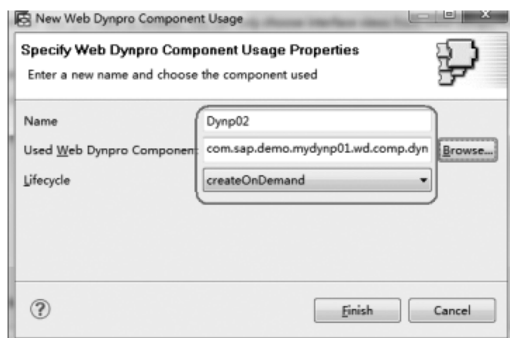


图 3-119　输入引用组件的名称

15）选择引用组件中的接口视图（见图 3-120），并单击“Finish”按钮。完成所有配置后，“Dynp01Comp”组件中新建的窗口中嵌入了一个接口视图，这个接口视图来源于刚刚新建的组件。

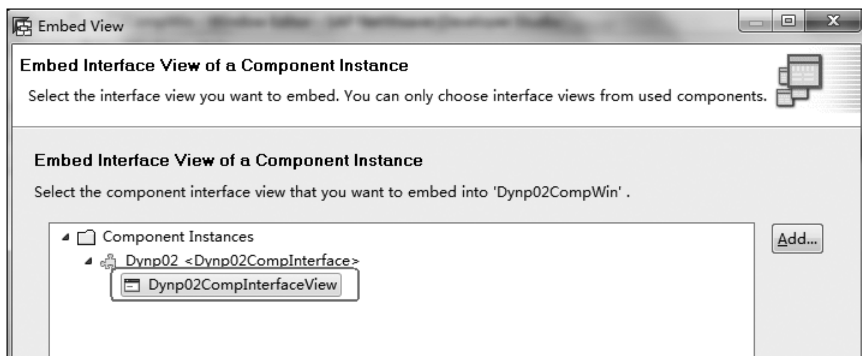


图 3-120　选择引用接口视图

16）为了能够清晰地区分两个组件，右击“Dynp02CompWin”，在弹出的快捷菜单中选择“Refactor”→“Rename”命令，将组件重命名为“Dynp01DisplayWin”（如图 3-121 所示）。

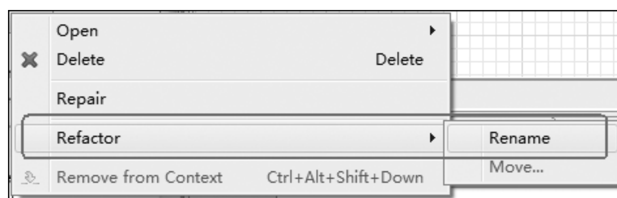


图 3-121　选择重命名

17）在弹出的对话框中输入自定义的名称，单击“Finish”按钮，如图 3-122 所示。

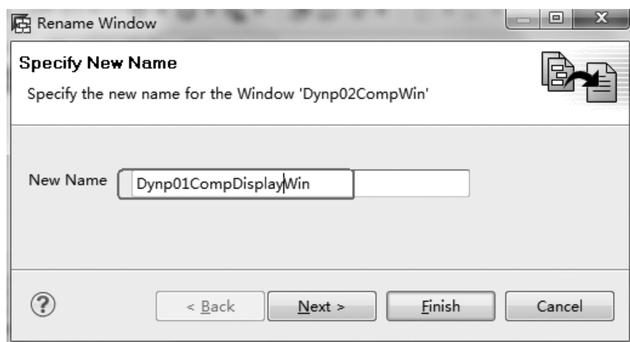


图 3-122　输入重命名名称

18）将接口视图“Dynp02IntV”重命名为“Dynp01Comp02IntV”，以方便区分。

19）下面简单编辑一下 Dynp02Comp 中视图的内容，打开视图，在视图中添加一个子元素，并选择“Caption”标题类型（见图 3-123），单击“OK”按钮。

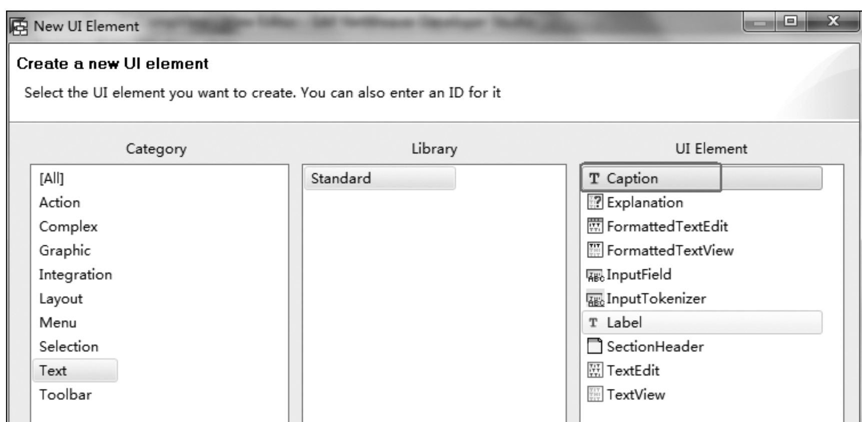


图 3-123　插入文本标题

20）将“Text”属性修改为“Dynp02Comp View Text”，如图 3-124 所示。

21）最后创建一个指向“Dynp01Comp02IntV”接口视图的应用。右击“Applications”，在弹出的快捷菜单中选择“Create Application”命令，自定义应用名称为“Dynp02Comp02App”，单击“Next”按钮，如图 3-125 所示。

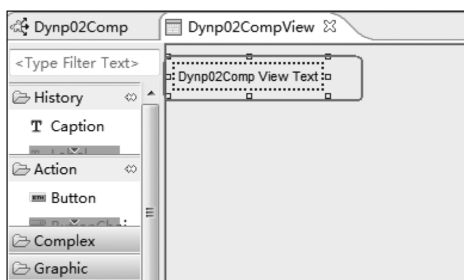


图 3-124　修改文本属性效果

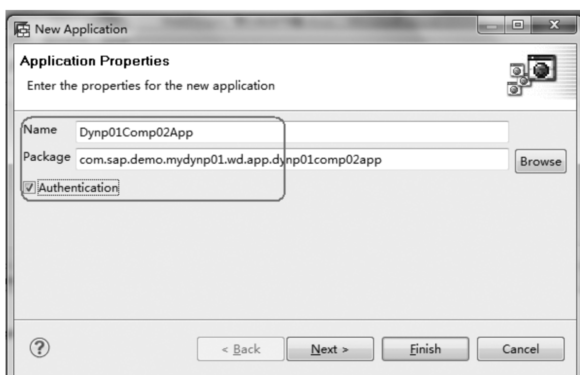


图 3-125　输入应用名称

22）选中“Use Existing Component”单选按钮，使用已存在的组件，单击“Next”按钮，如图 3-126 所示。

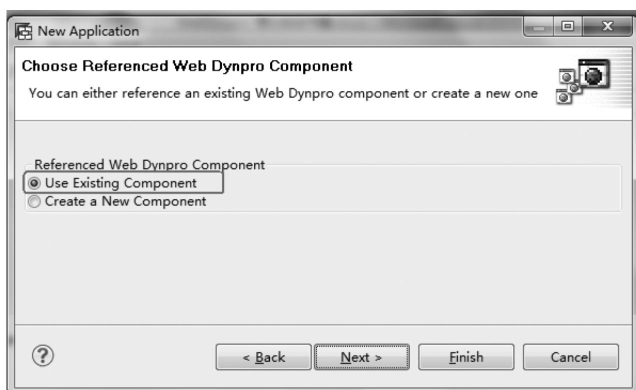


图 3-126　选择组件类型

23）随后选择“Dynp01Comp”，同时在接口视图中选择“Dynp01Comp02IntV”，单击“Finish”按钮，如图 3-127 所示。

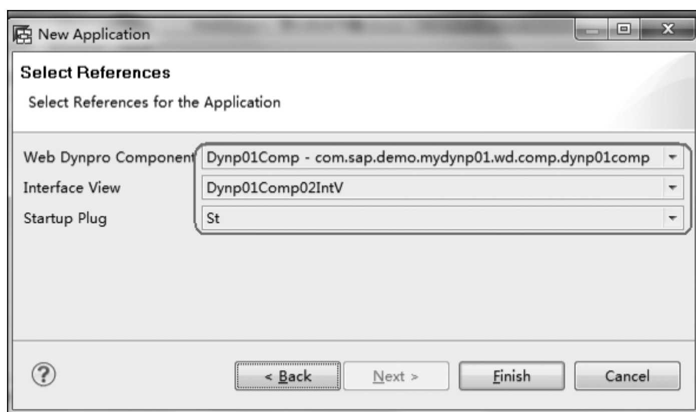


图 3-127　选择对应的接口视图

24）完成所有的对象配置后，在模型编辑器中可以看到相互之间的引用关系，如图 3-128 所示。其中“Dynp01Com02App”应用程序指向“Dynp01Comp”组件中指定的接口视图“Dynp01Comp02IntV”，而实现该接口视图的窗口“Dynp02CompDisplayWin”嵌入了“Dynp02Comp”组件中指定的接口视图，最终通过运行应用程序，展示了第二个组件的视图界面。

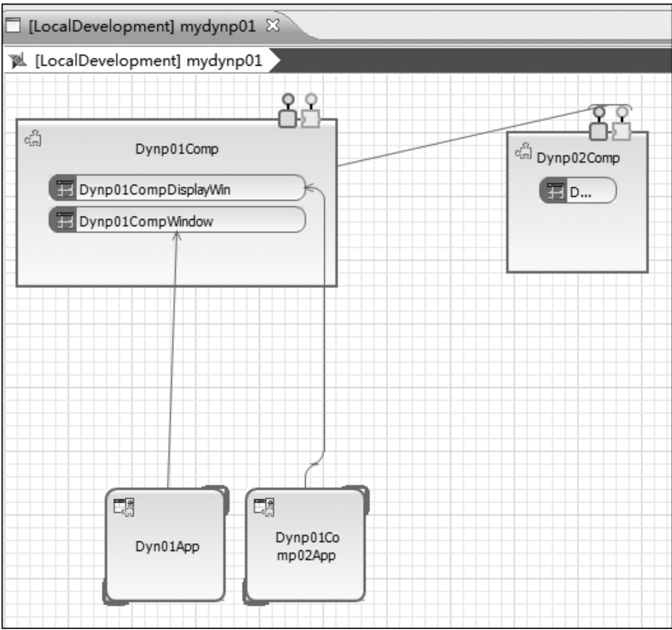


图 3-128　应用与组件的引用关系

25）保存所有修改，并重新编译，运行。右击“Dynp01Comp02App”，在弹出的快捷菜单中选择“Run”进行运行测试，如图 3-129 所示。

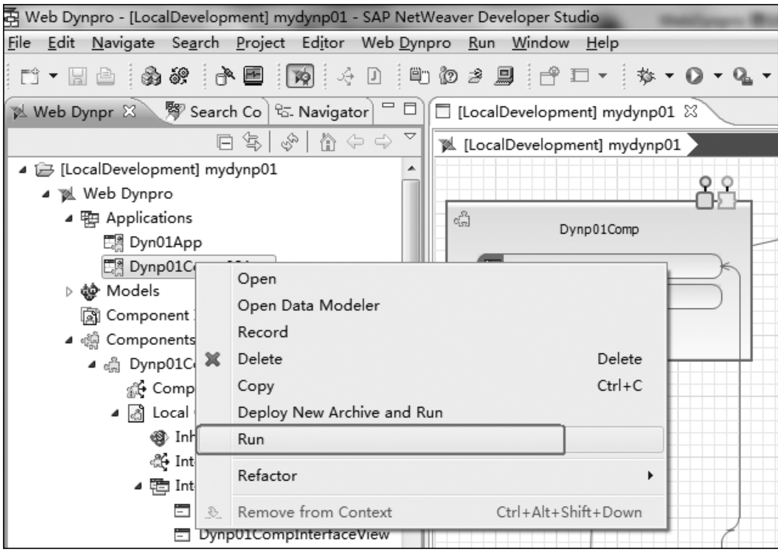


图 3-129　运行应用的菜单选择

26）运行结果显示了第二个组件中视图的内容，如图 3-130 所示。

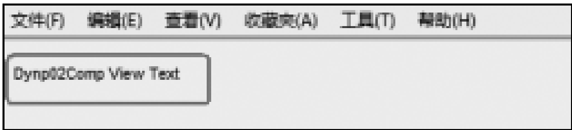


图 3-130 运行效果

3.3.3 视图集与视图嵌入

窗口中嵌入视图有以下几个特点。

- 嵌入的方式包括以下 3 种：创建一个新视图、插入一个已创建的视图、通过依赖的方式添加其他项目的接口视图。
- 对于通过任一形式被嵌入的 View 组件，可以设置其在 Window 是否默认显示的属性，当默认显示设置为 true 时，Window 初始化时会首先显示该 View 组件。

开发实例：

1）双击默认创建的窗口，右侧默认打开窗口模型编辑器。窗口中已嵌入了“Dynp01CompView”和“Dynp02CompView”两个视图，其中 02 视图的默认显示属性设置为“true”，所以在窗口的初始页面中将默认显示 02 视图的页面内容，如图 3-131 所示。

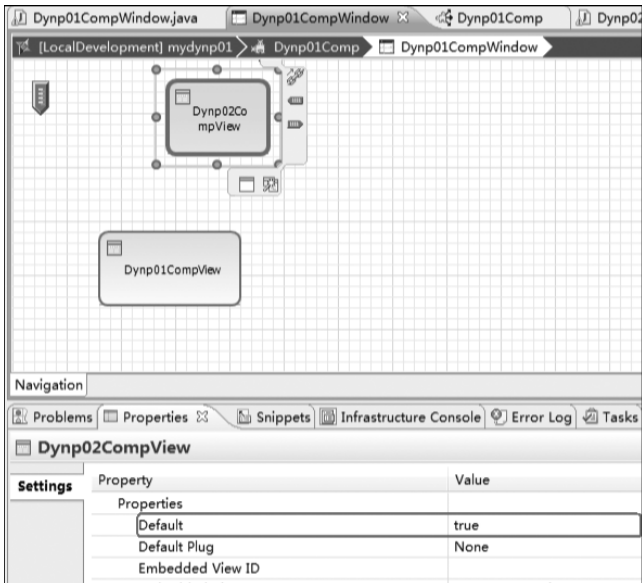


图 3-131 修改窗口中视图的属性

2）右击空白处，在弹出的快捷菜单中选择“Create ViewSet”命令，在弹出的对话框中输入自定义的视图集名称，并在下方选择视图集的布局，如图 3-132 所示。

3）选中新建的视图集，将属性中的“columns”设置为“1”，如图 3-133 所示。

4）右击“cell1”单元格，在弹出的快捷菜单中选择“Embed View”嵌入视图，选中“Embed Existing View”单选按钮嵌入已存在的视图（见图 3-134），单击“Next”按钮。

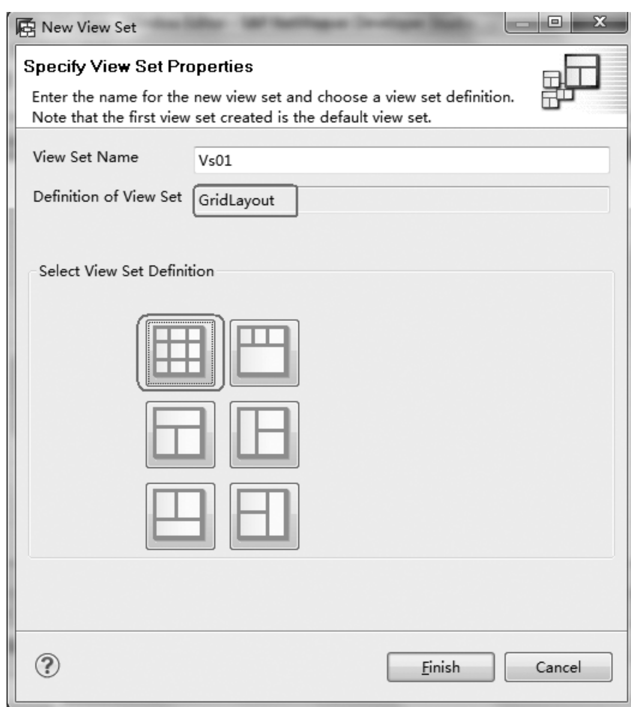


图 3-132　选择视图集的布局

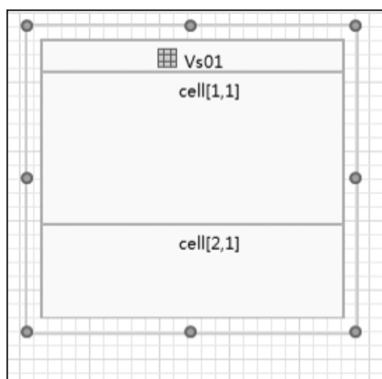


图 3-133　修改视图集的列数

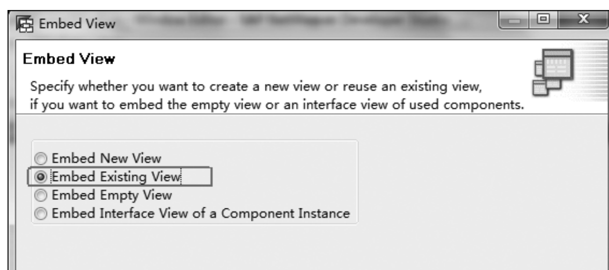


图 3-134　选择嵌入视图类型

5）选择“Dynp01CompView”（见图 3-135），单击“Finish”按钮。

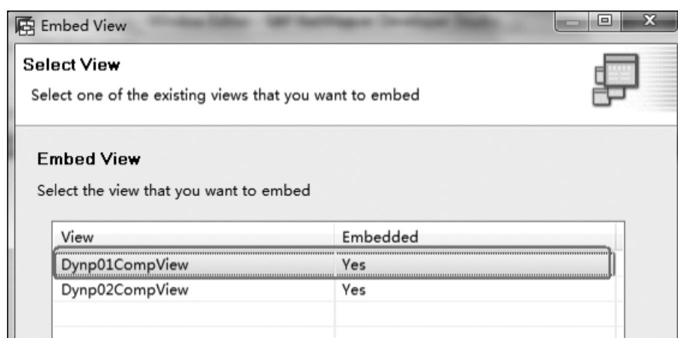


图 3-135　选择嵌入视图

6）重复刚才的操作，在下方的单元格中嵌入“Dynp02CompView”。

7）单击选中新建的视图集将默认显示属性修改为“true”，如图 3-136 所示。

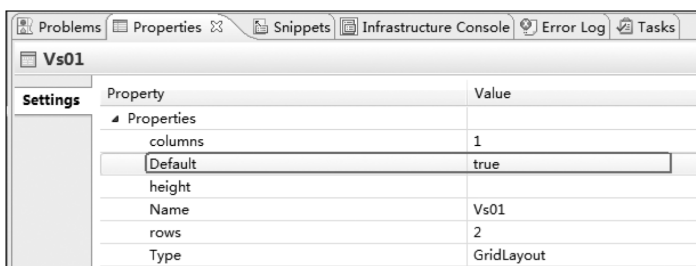


图 3-136　修改属性

8）保存所有修改，并重新编译，运行之后，即可进行相关测试，运行效果如图 3-137 所示。

注：在窗口中的一块显示区域内，每次仅能显示一个视图，而视图集中每一个单元格都是一块独立的显示区域。



图 3-137　运行效果

3.3.4　窗口的导航

在 3.1.5 节中，通过 Web Dynpro 创建向导的方式自动地创建了视图的导航并生成对应的按钮处理事件，下面对视图导航进行手动配置。开发过程包括了创建插口、配置导航连接和完成逻辑代码等步骤。

开发实例：

1）以上一小节的 Web Dynpro 项目为例，继续进行开发。首先在“Properties”中找到“Plugs”子标签，如图 3-138 所示。

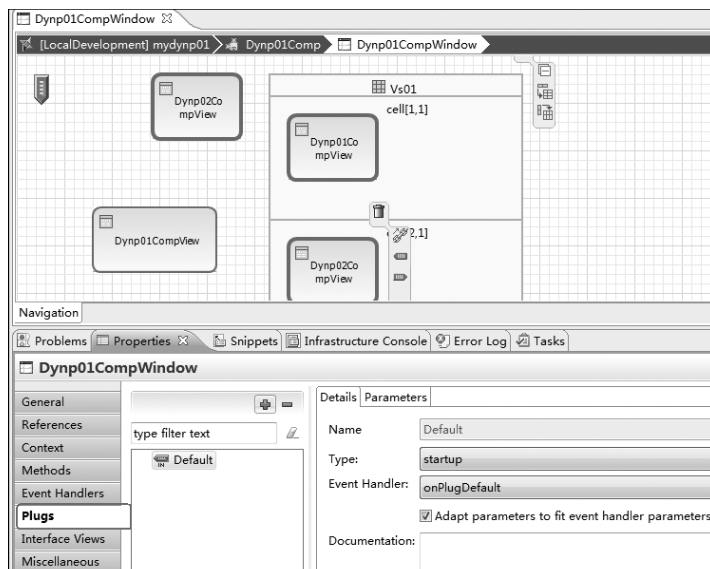


图 3-138　查看插口属性

2）单击加号，并选择“Outbound Plug”添加一个出口类型插口，如图 3-139 所示。

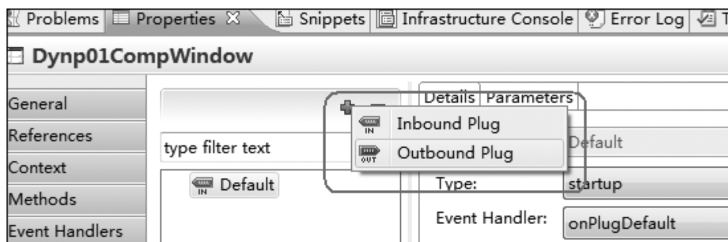


图 3-139　创建新的插口

3）输入插口名称“Out01”，单击“Finish”按钮，如图 3-140 所示。

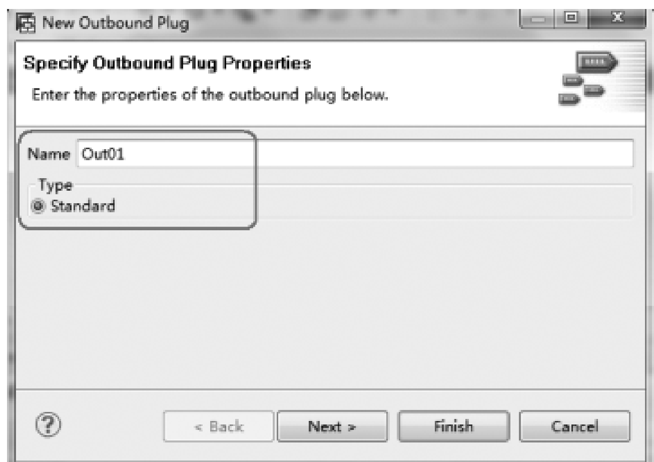


图 3-140　输入插口名称

4）使用相同的操作创建“Out02”插口。

5）接着创建“Inbound”类型插口，鼠标悬停在“Dynp02CompView”视图上，在弹出的菜单中选择蓝色箭头按钮，创建一个“Inbound”类型的插口，如图 3-141 所示。

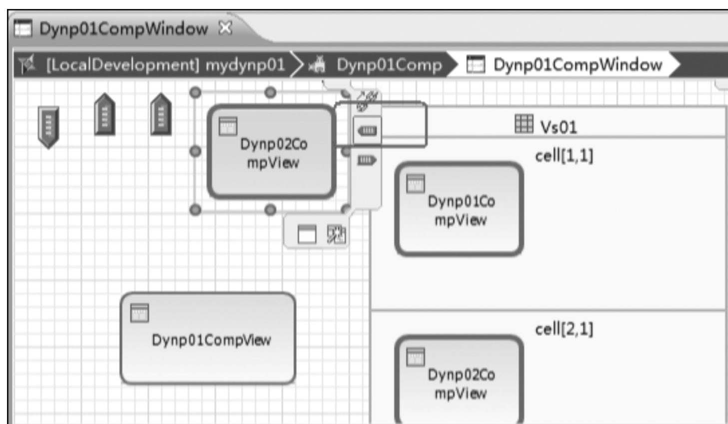


图 3-141　通过弹出菜单创建

6）按照上面的操作创建“Dynp01CompView”视图中的“Inbound”类型插口。

7）定义插口之间的导航连接，单击右边工具栏中的“Link”，如图 3-142 所示。

8）选中“Out01”并拖曳连线至“Dynp01CompView”视图的“Inbound”插口，完成导航连接的创建，如图 3-143 所示。

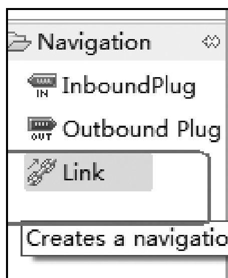


图 3-142　工具栏中的按钮

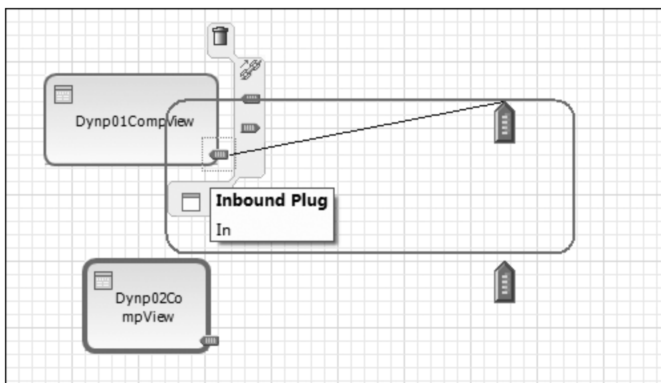


图 3-143　创建视图导航

9）同上完成从“Out02”到“Dynp02CompView”视图的关联。

10）切换到窗口的 Java 编辑器，在“wdDoInit( )”方法中添加以下代码：

```
wdThis. wdFirePlugOut01( ) ;
```

11）保存所有修改，重新编译、运行之后，即可进行相关测试。运行后，默认显示的是“Dynp01CompView”视图的界面（Hello World 应用），如图 3-144 所示。

12）接着修改“wdDoInit( )”方法的逻辑：

```
wdThis. wdFirePlugOut02( ) ;
```

13）保存所有修改，重新编译、运行之后，再次运行显示的内容变为“Dynp02CompView”视图界面，如图 3-145 所示。

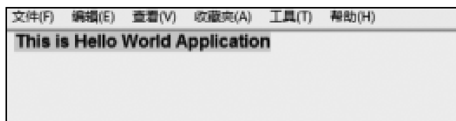


图 3-144　运行效果 1

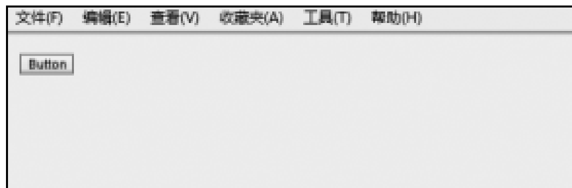


图 3-145　运行效果 2

3.3.5　其他窗口类型

除了 Web Dynpro 中标准的窗口以外，还可以通过代码定义和调用其他 3 种窗口类型，包括弹出窗口、外部窗口和确认窗口。

1. 弹出窗口

1）创建一个 Web Dynpro 项目“Dynp05”，并创建默认的组件、窗口和视图。随后创建一个新的窗口“Popup_Win_01”，如图 3-146 所示。

2）接着创建一个视图“Dynp05_01Comp_02View”，如图 3–147 所示。

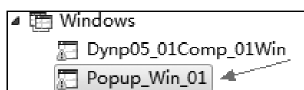


图 3–146　创建新窗口

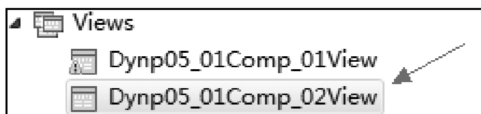


图 3–147　创建新视图

3）将创建的视图嵌入到新创建的“Popup_Win_01”窗口中，如图 3–148 所示。

4）组件控制器上下文中创建一个“Win_Info”结点，并创建一个“Win_Ins”的特性，用于保存和传递弹出窗口的实例对象，如图 3–149 所示。



图 3–148　嵌入对应视图



图 3–149　创建上下文特性

5）“Win_Info”结点的集合属性为“1・・1”，“Win_Ins”的类型为“IWDWindow”，如图 3–150 所示。在组件控制器中创建之结点后，将结点复制到两个视图中，并添加上下文的映射。

Type	com.sap.tc.webdynpro.services.session.api.IWDWindow

图 3–150　修改特性类型

6）在原有默认视图“Dynp05_01Comp_01View”中添加调用弹出窗口的按钮，如图 3–151 所示。

7）“明细”按钮处理事件的逻辑代码如下：

```
IWDWindow createModalWindow = wdComponentAPI. getWindowManager( ). createModalWindow( wd-
ControllerAPI. getComponent( ). getComponentInfo( ). findInWindows( "Popup_Win_01" ) ) ;
createModalWindow. setWindowSize( 200, 200) ;          //定义 Window Size
createModalWindow. setWindowPosition( 500, 100) ;      //定义 Window Position
createModalWindow. setTitle( "明细" ) ;
createModalWindow. show( ) ;
//保存 window 实例
IWin_InfoNode nodeWinInfo = wdThis. wdGetDynp05_01CompController( ). wdGetContext( ). nodeWin
_Info( ) ;
nodeWinInfo. currentWin_InfoElement( ). setWin_Ins( createModalWindow) ;
```

8）在新建的“Dynp05_01Comp_02View”视图中添加一些表单属性以及一个关闭按钮，如图 3–152 所示。

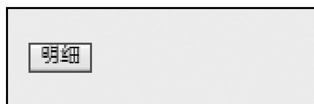


图 3–151　添加按钮效果



图 3–152　添加表单效果

9）“明细”按钮处理事件的逻辑代码如下：

```
//通过 Component Controller 上下文获取 Window 实例
IWDWindow winIns = wdThis. wdGetDynp05_01CompController( ). wdGetContext( )
. nodeWin_Info( ). currentWin_InfoElement( ). getWin_Ins( );
//关闭窗口
winIns. destroyInstance( );
```

10）保存所有修改，重新编译运行之后，即可进行相关测试，运行效果如图 3-153 所示。默认看到的是视图 01。

11）单击“明细”按钮，即可看到弹出窗口，如图 3-154 所示。



图 3-153　运行效果

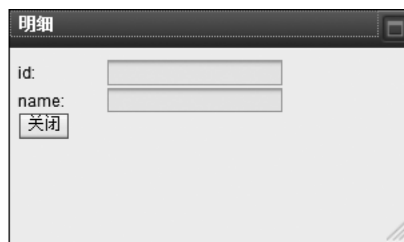


图 3-154　单击“明细”按钮后的运行效果

2. 外部窗口

外部窗口是指通过打开新的浏览器窗口，在新的浏览器窗口中显示 URL 链接对应的内容，不同于 Web Dynpro 的标准窗口。

开发实例：

1）继续使用之前创建的 Dynp05 项目，在视图 01 中添加一个按钮“Popup External Window”，如图 3-155 所示。



图 3-155　添加按钮效果

2）编写按钮对应事件的处理逻辑：

```
Map map = new HashMap( );                              //构造请求参数
map. put( "ID" , "C0010" );
String queryString = "" ;
String url = "" ;
try{
   queryString = WDURLGenerator. getQueryString( map, "UTF - 8" );  //根据 Map 转化为 Query-
//String
   url = WDURLGenerator. getApplicationURL( "demo. sap. com/mydynp05" ,
   "Dynp05_01Comp_02App" );                            //根据 DeployObject 与
   // Application 获取标准的 ApplicationURL
} catch( Exception e) {
   // TODO Auto - generated catch block
   // e. printStackTrace( );
   wdComponentAPI. getMessageManager( ). reportException( e. getMessage( ) );
   logger. errorT( e. getMessage( ) );
   return;
}
IWDWindow createNonModalExternalWindow = wdComponentAPI
. getWindowManager( ). createNonModalExternalWindow( url,
```

```
queryString, true);                       //创建 External Window,请求访问的方式为 POST
createNonModalExternalWindow. show();     //窗口显示
```

3）在视图 02 中的“wdDoInit”初始化方法中打印参数标识，添加代码如下：

```
String parameter =
  WDProtocolAdapter. getProtocolAdapter(). getRequestObject(). getParameter("ID");//获取 Pa-
//rameter ID Value
  wdComponentAPI. getMessageManager(). reportSuccess(parameter);
```

4）保存所有修改，重新编译、运行之后，即可进行相关测试，运行效果如图 3-156 所示。



图 3-156 运行效果

5）单击“Popup External Window”按钮，弹出的明细页面打印了参数的标识，如图 3-157 所示。

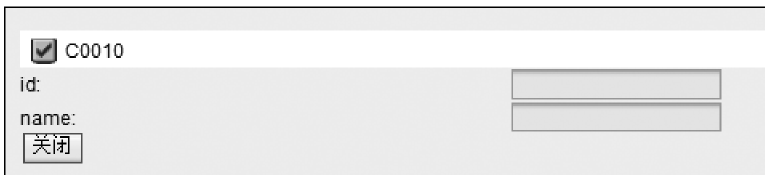


图 3-157 运行效果

3. 确认窗口

确认窗是 Web Dynpro 中封装过的窗口（对话框），通过 API 调用可直接弹出确认窗口，确认窗口的确认与取消的逻辑自定义。

开发实例：

1）继续使用的之前视图，添加一个“Popup Confirmation Window”按钮，如图 3-158 所示。



图 3-158 添加按钮效果

2）编写按钮逻辑，代码如下：

```
//创建 Dialog Window,默认添加的确认选项
IWDConfirmationDialog createConfirmationWindow = wdComponentAPI. getWindowManager ( )
. createConfirmationWindow("是否确认?",wdThis. wdGetOkAction(), "确认");
//添加取消的选项
createConfirmationWindow. addChoice(wdThis. wdGetCancelAction(), "取消");
//弹出 Window
createConfirmationWindow. show();
```

3）创建确认对话框中需要使用的确认与取消的自定义事件“Ok”与“Cancel”，如图 3-159 所示。

4）编写确认逻辑，打印成功提醒消息：

```
wdComponentAPI. wdGetMessageManager( ). reportSuccess("OK");
```

5）保存所有修改，重新编译，运行之后，即可进行相关测试，运行效果如图 3-160 所示。

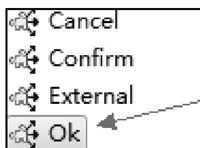


图 3-159　添加自定义方法



图 3-160　运行效果

6）单击“Popup Confirmation Window”即可看到弹出的“确认”对话框，如图 3-161 所示。

7）单击“确认”按钮，窗口关闭并出现打印成功提醒消息，如图 3-162 所示。

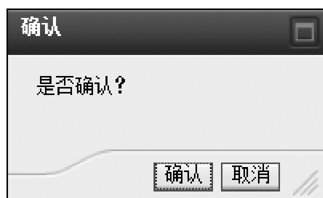


图 3-161　单击按钮效果



图 3-162　单击“确认”按钮后的效果

3.4　组件控制器

如果说之前的视图和窗口的介绍还可以算是“MVC”中“V”的部分，甚至熟练的开发人员不需要编写任何代码就能完成“V”的一些简单设计，本节的组件控制器（Component Controller）则来到了“MVC”中的“C”部分，将会不可避免地涉及 Web Dynpro 代码的编写。

组件控制器会在创建组件时自动生成，作为整个组件的主控制器对数据进行处理。简单地说，视图窗口是 Web Dynpro 对应组件的脸，而组件控制器则是组件的大脑。

在 Web Dynpro 设计模式中，视图、窗口以及其他控制器都可以通过与组件控制器的上下文映射（Context Mapping）实现数据共享，组件控制器在逻辑处理、数据传递中起到了控制与转换的作用；同时组件控制器或自定义控制器（Custom Controller）直接与模型数据源交互，并将处理后的数据传递给视图，最终成为与用户直接交互的 UI 层数据。

3.4.1　组件控制器的属性

从 Web Dynpro 工程的目录双击组件控制器，会自动打开对应的模型编辑器；如需编辑控制器源代码，则先右击组件控制器，在弹出的快捷菜单中选择“Open”→“Java Editor”即可打开 Java 编辑器查看源代码，如图 3-163 所示。

需要说明的是，虽然在操作时双击打开的是组件控制器，但如果仔细看会发现右上方打

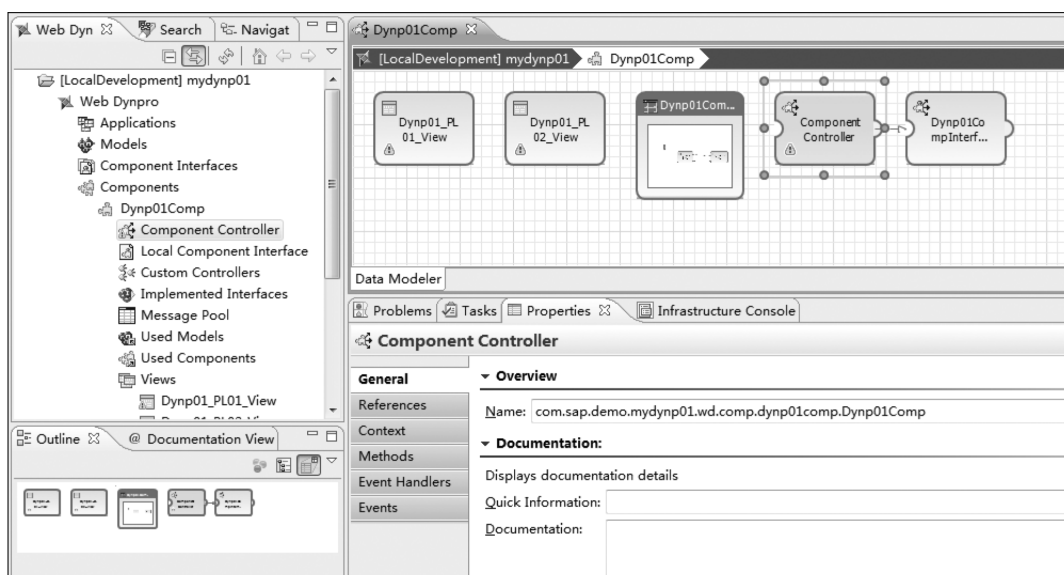


图 3-163　组件控制器的属性

开的模型编辑器其实是整个组件（图 3-163 中为“Dynp01Comp”组件）的模型编辑器，如在编辑器中选中视图或其他对象，则右下方的“Properties”下也会切换显示为相应对象的所有属性。

组件控制器中包含的属性子标签及功能说明见表 3-5。

表 3-5　组件控制器中包含的属性子标签及功能说明

属性子标签	功 能 说 明
General	组件控制器的一般信息
References	其他组件的引用列表
Context	组件控制器的上下文
Methods	组件控制器的自定义方法
Event Handlers	组件控制器的事件处理器
Events	自定义事件

3.4.2　组件控制器的生命周期

组件控制器的生命周期与其对应的组件实例的生命周期是一致的，甚至大部分情况下，可以说组件控制器的生命周期决定了整个组件的存在，即组件控制器被调用并实例化，同时创建相应组件的实例，最后在控制器退出时销毁组件实例。以下是整个组件控制器的生命周期中可能会执行到的标准方法。

- wdDoInit()：这是所有控制器（包括组件控制器和视图控制器）的默认初始化方法，且在实例初始化时仅执行一次，因此对于控制器中数据的初始化或者变量的默认值设定都在这个方法中完成。
- wdDoExit()：与 wdDoInit()相对，该方法在每个控制器结束时执行。视图的退出场景

有很多，包括视图失效、窗口关闭、视图切换等。该方法也是每个控制器实例的出口。

- wdDoPostProcessing( )：该方法仅在组件控制器中存在。由于在复杂的 Web Dynpro 应用中（如跨组件或跨工程的方法调用），数据可能来自于多个组件，因此为了程序更有效地运行，该方法可以实现相关代码，使其在数据处理之前就对数据提前进行有效性验证。
- wdDoBeforeNavigation( )：该方法也仅在组件控制器中存在。当一个视图控制器触发一个 outbound 的插口时，它并不是一个人在战斗，很可能有许多视图控制器都触发了视图切换的动作，这些请求随后会进入一个处理队列，而这个方法可以实现相关代码，对队列中导航之前发生的逻辑动作进行处理（如导航事件优先级）。
- isCurrentRoot 布尔参数：由于上述两个方法会涉及多个组件的交互，而仅有 Web Dynpro 运行时默认显示的接口视图对应所在的组件才能被称为根组件（root component），因此该参数用于区分此类组件。

组件中视图、窗口、组件控制器的标准初始化方法执行顺序如下。

1）组件控制器的初始化：“Component Controller” 的 “wdDoInit” 方法。

2）默认窗口的初始化：“Window” 的 “wdDoInit” 方法。

3）默认视图的初始化：“View” 的 “wdDoInit” 方法。

4）视图的默认校验方法：“View” 的 “wdDoBeforeAction” 方法。

5）视图 UI 元素对应的方法：自定义 “Action”。

6）视图的默认变更方法：“View” 的 “wdDoModifyView” 方法。

注：更详细的视图标准方法执行顺序请参考 3.1.1 节的开发实例。

开发实例：

1）切换到组件控制器 wdDoInit( )，右击 “Dynp01Comp Component Controller”，在弹出的快捷菜单中选择 “Open” → “Java Editor”，并添加以下代码，如图 3-164 所示。

```
wdComponentAPI.getMessageManager().reportSuccess("Executer Component Controller wdDoInit():");
```

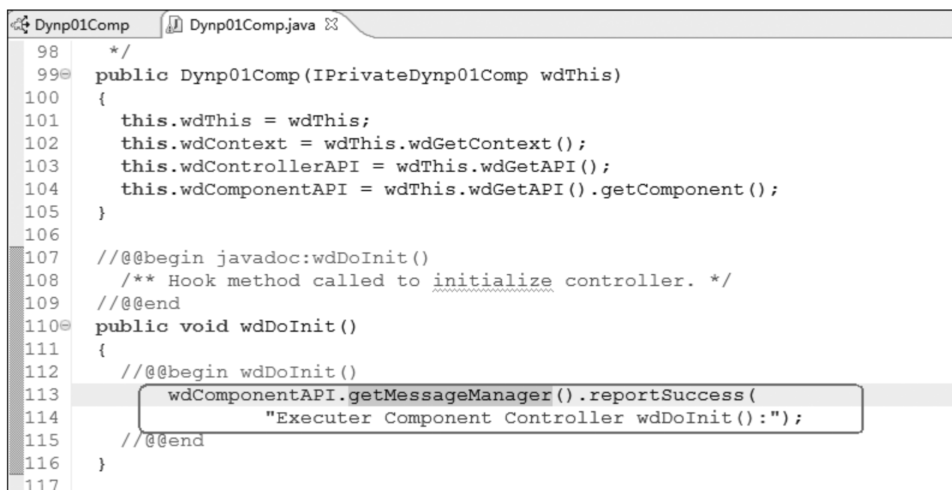


图 3-164　添加代码

2）保存所有修改，重新编译、运行之后，即可进行“Dynp01App”应用的运行测试，运行效果如图 3-165 所示。



图 3-165　运行效果

通过图 3-165 的运行结果可以看到，当程序运行并创建一个组件的实例之后，会先执行组件控制器初始化方法，再执行窗口的初始化，最后执行视图的初始化。

3.4.3　组件控制器的标准类

与视图控制器创建时类似（参照 3.1.2 节），组件控制器创建的同时自动生成的标准类包括 IPublic <NAME> Comp、IPrivate <NAME> Comp、Internal <NAME> Comp 和 <NAME> Comp。

组件控制器的标准类默认创建目录如下：

```
<NWDS directory>\workspace.jdi\LocalDevelopment\DCs\
    <package name>\<project name>\_comp\gen_wdp\packages\
    <package directory>\wd\comp\<component name>\wdp
```

组件控制器相关标准类和继承关系如图 3-166 所示。

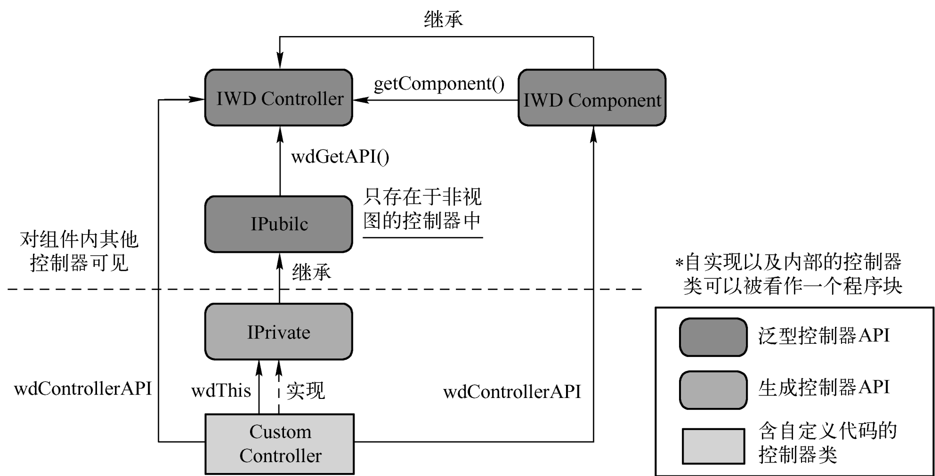


图 3-166　组件控制器相关标准类和继承关系

以之前的开发实例 mydynp01 为例，对应的组件控制器的标准类的目录如下：

..\workspace.jdi\LocalDevelopment\DCs\demo.sap.com\mydynp01_comp\gen_wdp\packages\com\sap\demo\mydynp01\wd\comp\dynp01comp\wdp

- IPublic <NAME> Comp：包含所有组件控制器相关 API 以及数据结构定义，代码自动生成。
- IPrivate <NAME> Comp：继承了 IPublic <NAME> Comp 接口，也是组件控制器中 wdThis 对应的实例接口，代码自动生成。

- Internal<NAME>Comp：实现 IPrivate<NAME>Comp 接口，同时继承了 GCIComponentController 标准类，并实现了 IGCIComponentDelegate 标准接口，代码自动生成。
- <NAME>Comp：Web Dynpro 为了更好地实现对标准程序与自定义程序的剥离，Internal<NAME>Comp 采用了另外一种设计模式，在初始化时会生成包含所有可自定义方法的<NAME>Comp 类，开发者在编辑组件控制器时，不是在编辑 Internal<NAME>Comp 而是在编辑<NAME>Comp 类实例。

3.4.4 组件控制器的上下文

组件控制器的上下文（Context）是对组件控制器数据结构的定义，所有上下文中存放的数据都是私有的，只有在与其他组件或控制器完成上下文映射或者对外声明以后，才能被其他组件使用。上下文的基本介绍、属性以及操作的 API 都与 3.1 节中视图的上下文一致，所以本节不再重复介绍上下文的概念，重点从组件控制器的实际应用进行介绍。

从设计的角度来讲，视图的上下文用来绑定视图的页面元素和进行属性控制，自定义控制器的上下文用来与模型进行数据交互，而组件控制器的上下文更多的是用于中转与控制，而并非是组件的数据源。上下文可以作为控制器自身的一个小型本地数据库，但真正的数据源更多还是来自于各种各样的模型，组件控制器的上下文只有在结合了 UI 层的数据访问或者后台模型的数据交换，才会体现意义，毕竟核心的逻辑处理还是需要在组件控制器中完成。

1. 上下文的映射

组件控制器的上下文在整个 Web Dynpro 工程的数据结构中起到控制与中转的作用，其他的组件可以通过组件控制器实现数据的共享与传递，上下文的映射就是其他组件直接与组件控制器作为数据共享的主要方式。

这种映射方式的优点：通过映射关系，所有相关的控制器都被允许使用来自于上下文的源数据；在这样的要求下仍然只需要维护一份数据，不同控制器中处理上下文的数据将通过映射关系进行同步；数据可以进行选择性地映射，避免冗余结构和字段的出现，使得其他控制器的运行更为有效。

要注意的是，视图控制器的上下文并不允许成为映射源，从 MVC 的架构来说，视图控制器的目的是管理视图的 UI 元素、导航链接事件等内容，而不是处理数据，所以一般来说都是以组件控制器的上下文为映射源；从技术角度来看，由于视图控制器不存在 IPublic<NAME>类型的公共接口，而 IPublic<NAME>接口的数据结构为公共外部类可访问的结构，因此视图控制器的数据结构具有受限制访问权限，不能作为映射源。

不同控制器之间相同结构类型的上下文做了映射之后，上下文中的数据保存在源结点中，所有对映射结点或源结点的操作都集中于源结点的数据，并通过这种方式达到不同控制器所使用数据的一致。控制器上下文的映射关系如图 3-167 所示。

图 3-167 中左侧的两个方框都是视图控制器，右侧分别为自定义控制器和组件控制器，映射箭头附件的字母“O”表示“Original”源结点，而字母“M”表示“Mapped”映射结点。通过图 3-167 可以看到：

1）不允许视图控制器之间进行上下文映射，因为视图控制器的上下文并不能获取源数据。

2）不允许自定义控制器指向视图控制器的上下文映射。

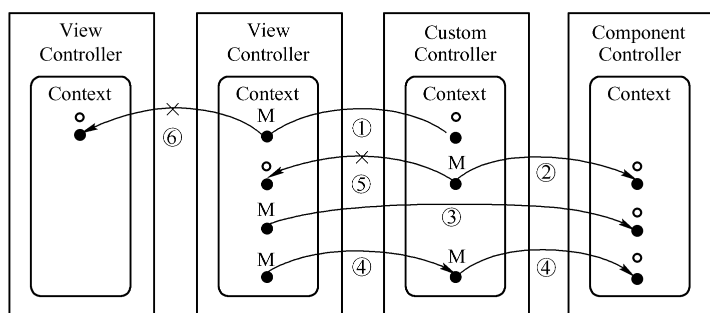


图 3-167　上下文映射关系

3）允许从视图控制器指向自定义控制器以及组件控制器的上下文映射关系。

4）允许从自定义控制器指向组件控制器的上下文映射关系。

2. 视图与组件控制器

对于上下文中结点的映射，映射结点使用源结点作为数据源，而映射结点的数据与源结点始终保持一致。上下文结点的映射有两种不同的方式，两个结点的属性也可以根据映射方式的不同会产生差异。两种映射方式分别为“Collection and selection”集合与选择属性、“Collection Only”仅集合属性，默认情况下映射类型为集合与选择属性。

1）集合与选择属性：这种方式代表映射结点与源结点的属性（包括集合属性、选择属性）完全一致，并以源结点为基准。

2）仅集合属性：这种方式代表映射结点与源结点的属性中只有集合属性保持一致，选择属性可以单独设置。

另外，对于视图与组件控制器的上下文映射的实际操作，简单来说依次是：在视图和组件控制器中分别创建上下文，建立引用关系，完成具体映射的配置。而在实际操作过程中，“建立引用关系”也有 3 种不同的配置方式来实现同样的效果：

1）在组件的模型设计器里通过数据链接的连线建立关系。在模型设计器中，单击“Data Link”数据链接，单击视图并拖曳至组件控制器，如图 3-168 所示。

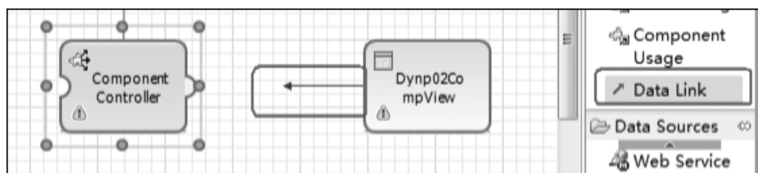


图 3-168　创建引用关系

2）在视图属性的“References”子标签中，添加对组件控制器的引用，如图 3-169 所示。

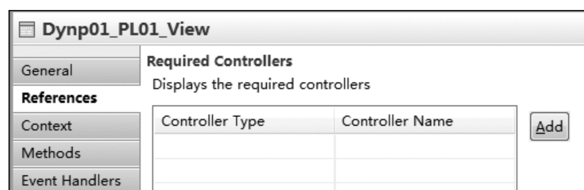


图 3-169　添加引用对象

3）在创建视图的上下文后，直接对其编辑映射关系：右击上下文结点，在弹出的快捷菜单中选择“Edit Context Mapping”，编辑上下文映射，如图 3-170 所示。

开发实例：

1）接下来在组件控制器上下文中完成一组数据定义，以“mydynp01”项目为例，配置并测试映射完成后的实际效果，如图 3-171 所示。

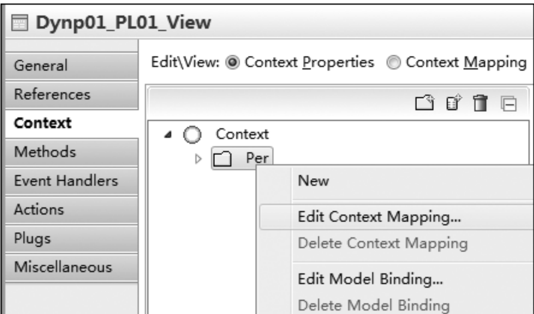


图 3-170　编辑上下文映射

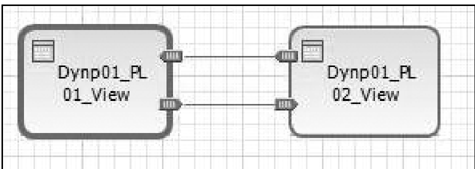


图 3-171　原 Web Dynpro 效果

2）切换到视图上下文界面，编辑“Per”结点，添加“string”字符串类型的 name 与 desc 两个特性，修改“id”元素的属性为不可计算，如图 3-172 所示。

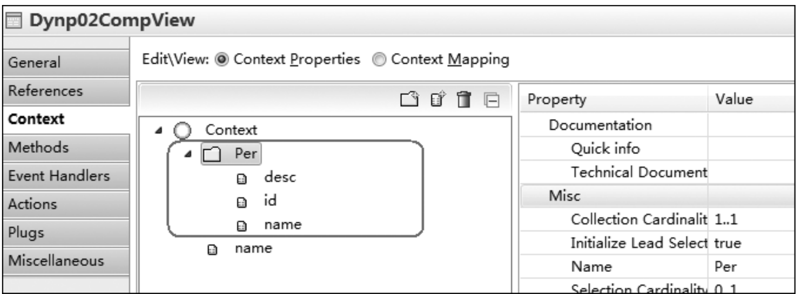


图 3-172　修改上下文属性

3）右击“Per”结点，在弹出的快捷菜单中选择“复制”，打开组件控制器的上下文，右击“Context”，在弹出的快捷菜单中选择“粘贴”，完成在组件控制器上下文中定义与视图控制器上下文结构一致的结点，如图 3-173 所示。

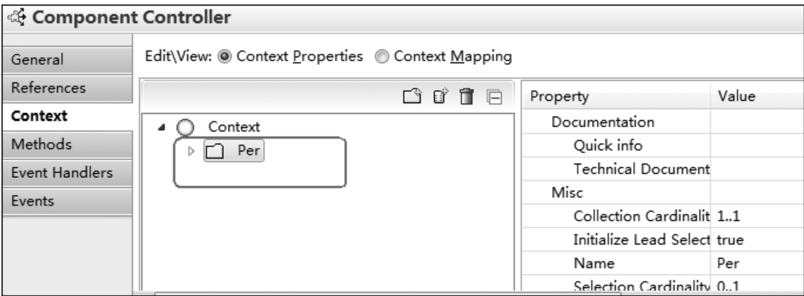


图 3-173　在组件控制器中创建上下文

4）双击组件控制器，打开组件的模型编辑视图，单击右侧的“Data Link”进行数据链接，然后单击“Dynp02CompView”并拖曳连线至“Component Controller”，如图 3－174 所示。

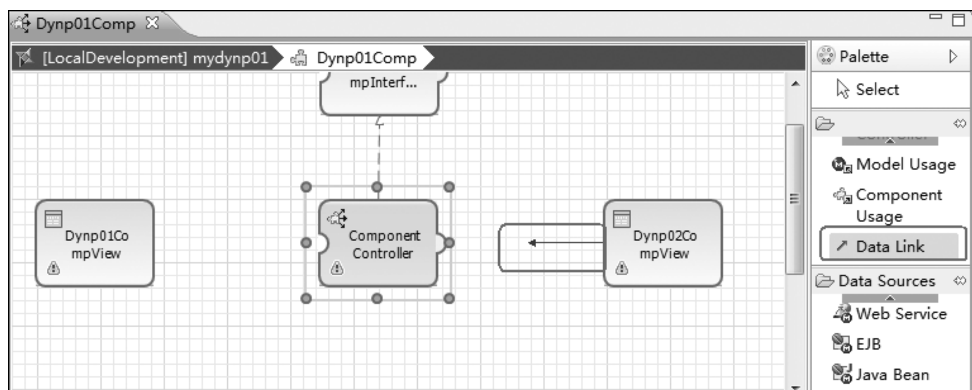


图 3-174　创建数据链接

5）拖曳后，自动弹出上下文映射窗口。左侧显示的是视图控制器的上下文，右侧显示的是组件控制器的上下文，上方还有文字提醒，如图 3-175 所示。如果先在组件控制器中定义上下文，则直接可以通过上下文映射时在视图控制器中创建对应上下文并完成上下文映射。

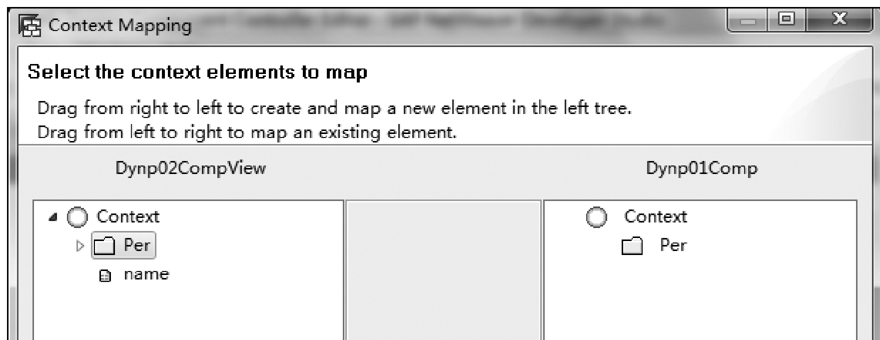


图 3-175　编辑上下文映射

6）单击左侧的“Per”结点并拖曳至右侧组件控制器的结点上，系统弹出详细映射规则定义的窗口，如图 3-176 所示。

7）选择希望做映射的特性，单击“Finish”按钮，如图 3-177 所示。

8）完成映射元素选择后，在上下文映射窗口中会显示已映射元素之间的连接线（见图 3-178），确认无误后单击“Finish”按钮。

9）再次切换到视图的上下文界面，“Per”结点的图标上已多了一个小箭头，表示已映射；同时它的属性中也已标记“Mapping Path”映射路径和“Mapping Type”映射类型，如图 3-179 所示。由于映射类型是“集合与选择属性”，因此该结点的集合属性与选择属性都不可修改。

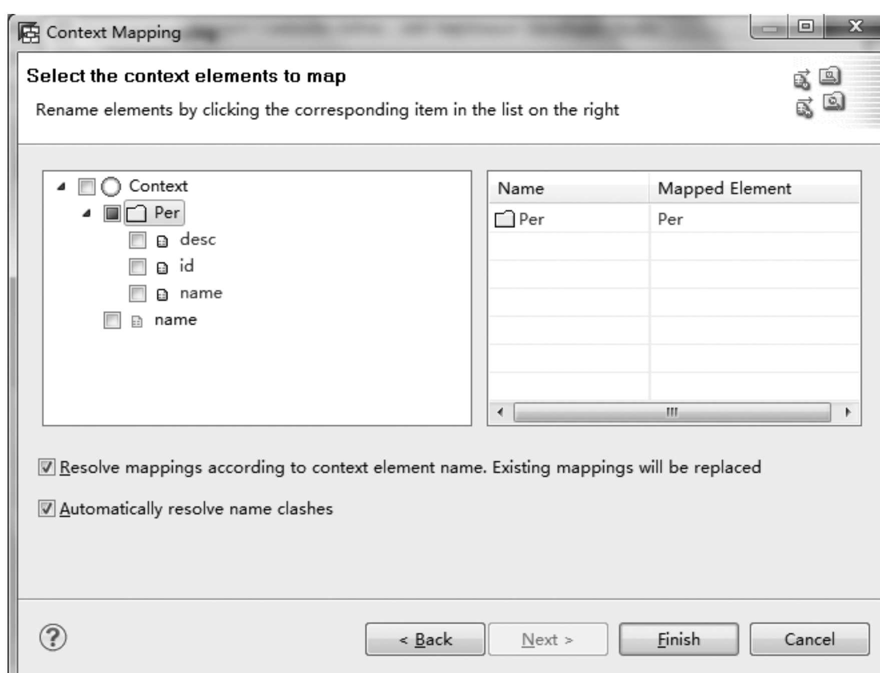


图 3-176　选择映射的上下文结点

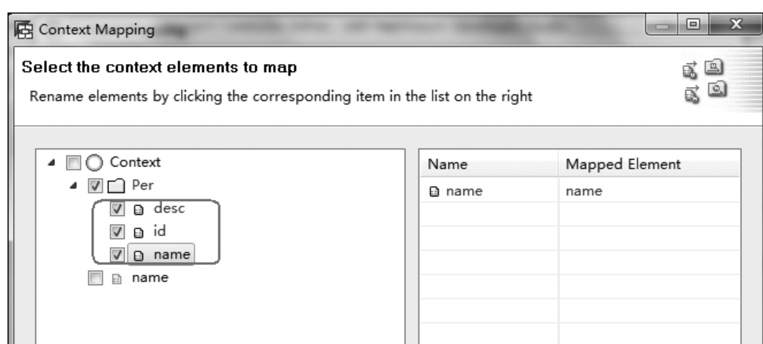


图 3-177　选择映射的上下文特性

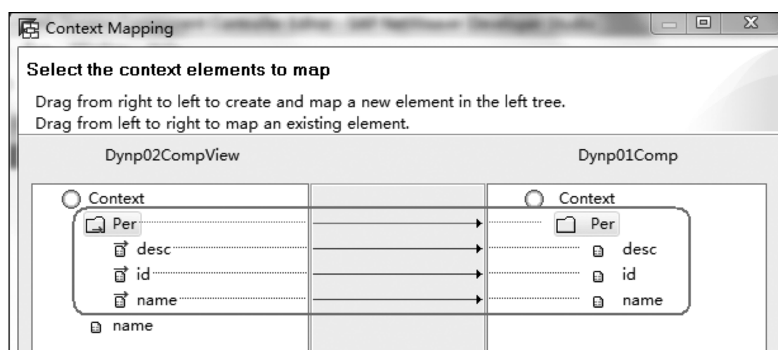


图 3-178　确认映射对象

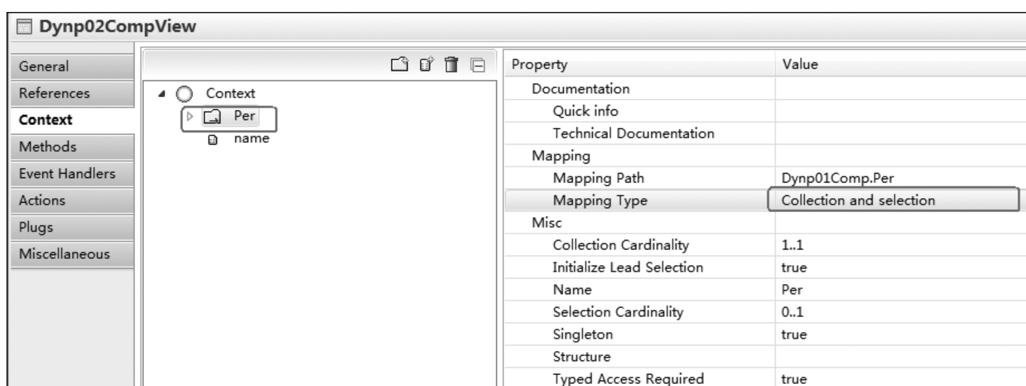


图 3-179　映射类型属性

10）下面进行修改测试。打开组件控制器上下文，分别修改“Per”结点的集合属性与选择属性，如图 3-180 所示。

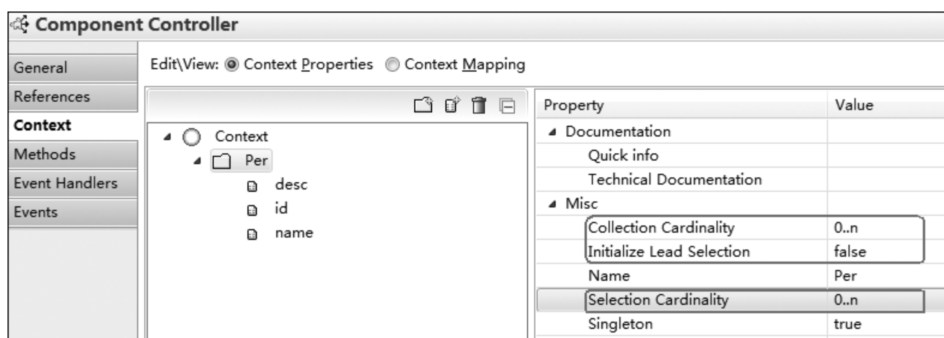


图 3-180　修改组件控制器中的上下文属性

11）在保存上述修改后，打开视图上下文进行查看，映射结点的属性也已同步修改，如图 3-181 所示。

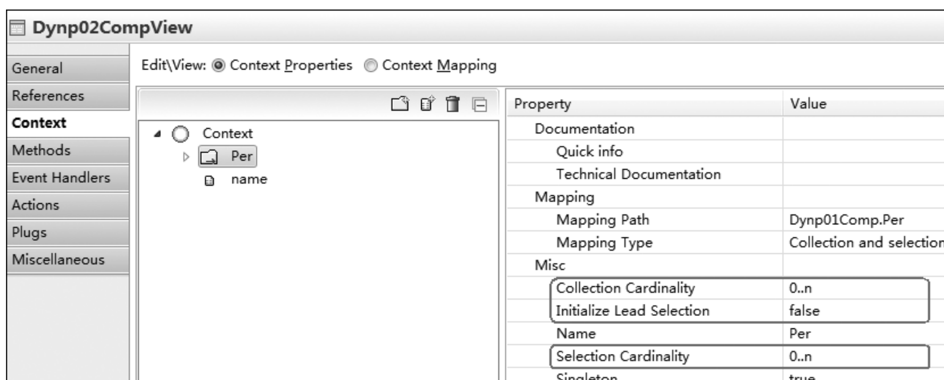


图 3-181　查看视图中的上下文属性

读者可以自行尝试将视图控制器上下文的映射类型修改为“Collection Only”仅集合属性，然后再修改组件控制器中的上下文属性，看看视图控制器中是否会同步发生改变。

3. 视图与视图

Web Dynpro 项目内组件之间的引用原理与类之间的引用非常相似，组件可以访问引用组件的数据与方法，组件中的 4 种控制器中，除了视图控制器不能被引用之外，其他控制器之间都可以相互引用。引用同时也是上下文映射的前提，视图作为终端访问组件，其上下文结点不能作为源结点，视图控制器的 Java 类内部也不建议有复杂的业务逻辑（除数据层的业务逻辑外，建议全部封装为组件控制器的操作单元），所以视图与视图之间、视图与其他组件之间的交互都是通过组件控制器来完成的，视图不允许被直接引用。

引用的方式包括以下两种：

- Data Link：直接通过 Data Link 使两个组件进行关联。
- References 属性：在 Properties 属性中，References 用来添加被引用的其他组件。

开发实例：

1）以 mydynp01 工程为例，之前完成的是视图间的切换，如果进一步地还需要在切换时传递参数，则需进行开发配置，如图 3-182 所示。

2）首先需要在 Dynp01_PL01_View 和 Dynp01_PL02_View 中定义数据存放的上下文，并且完成视图与控制器的映射，如图 3-183 所示。根据上一小节的配置，已完成了 PL02 视图的上下文映射，下面来完成 PL01 视图的上下文映射。首先将“Per”结点复制并粘贴到“Dynp01_PL01_View”视图的上下文中，右击“Per”结点，在弹出的快捷菜单中选择编辑上下文映射。

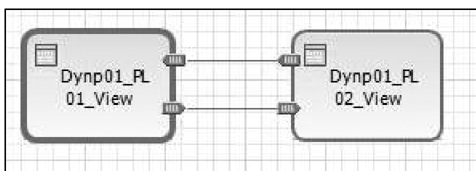


图 3-182　原 Web Dynpro 项目效果

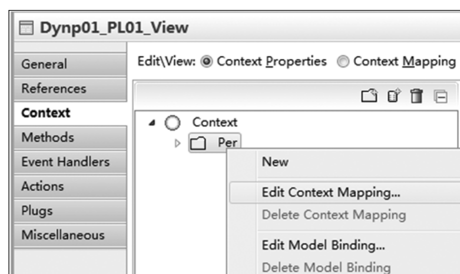


图 3-183　编辑上下文映射

3）由于是将视图控制器与组件控制器进行映射，因此在弹出的窗口中选中组件控制器，并单击“OK”按钮，如图 3-184 所示。

4）在弹出的窗口中，就可以看到组件控制器的上下文，同样选中“Per”结点（见图 3-185），单击“Next”按钮。

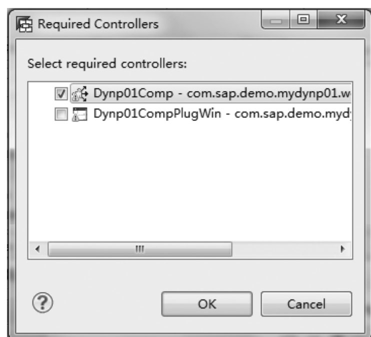


图 3-184　选择引用的对象

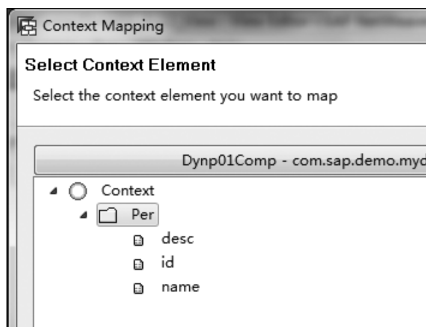


图 3-185　选择映射的结点

5）选中需要映射的特性字段（见图 3-186），单击“Finish”按钮。

6）保存之后，视图 PL01 的上下文映射也已经完成，如图 3-187 所示。

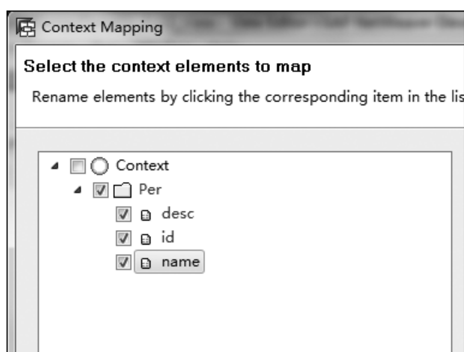


图 3-186　选择映射的特性

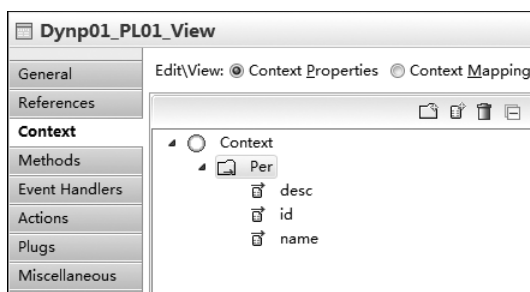


图 3-187　完成映射后的效果

7）在本开发实例中，不再通过消息打印来进行参数的判断，所以需要对视图添加页面的 UI 元素。右击“Outline”中的“RootElement”元素，在弹出的快捷菜单中选择“Apply Template”，如图 3-188 所示。

8）在弹出的窗口中选择“Form”表单形式，单击“Next”按钮，如图 3-189 所示。

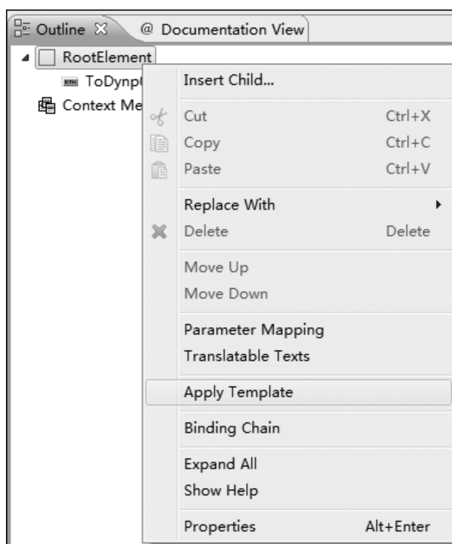


图 3-188　选择应用模板

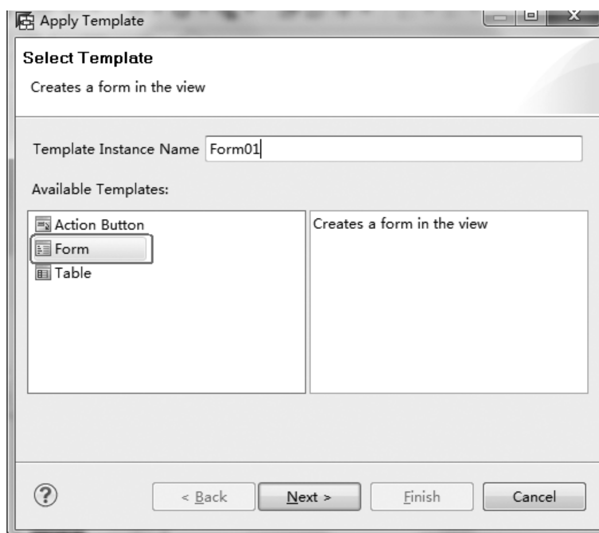


图 3-189　选择模板类型

9）选择表单需要绑定的上下文结点和特性，单击“Finish”按钮，如图 3-190 所示。

10）在弹出的窗口中，对字段顺序和页面元素进行配置，单击“Finish”按钮，如图 3-191 所示。在这个窗口中，可以修改页面 UI 元素类型，如使用文本框、下拉列表或单选按钮等。

11）保存之后，即自动生成了一个 Web Dynpro 页面，如图 3-192 所示。

12）使用相同的操作和配置，对 Dynp01_PL01_View 也生成对应的表单页面元素，如图 3-193 所示。

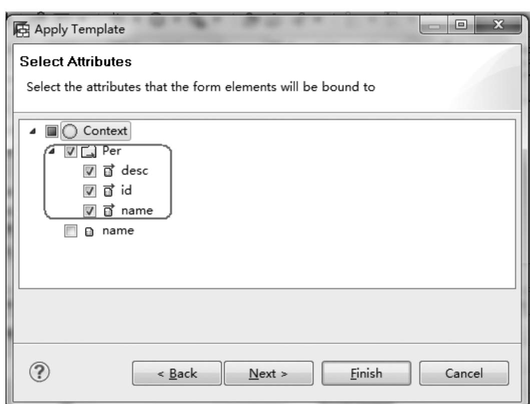


图 3-190　选择对应上下文

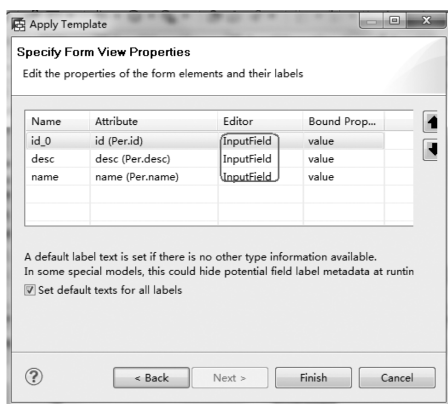


图 3-191　修改视图元素类型

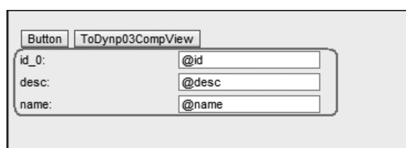


图 3-192　运行效果 1

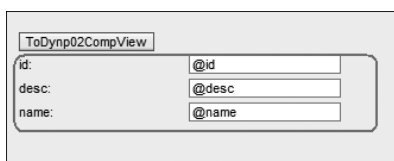


图 3-193　运行效果 2

13）编译、部署、运行（Dynp01Comp03App），在“id”“desc”和“name”文本框中输入任意内容，如图 3-194 所示。

14）单击“ToDynp02CompView”按钮并触发页面跳转至 PL02 视图，会发现第一个视图填写的数据成功传输到了第二个视图，如图 3-195 所示。

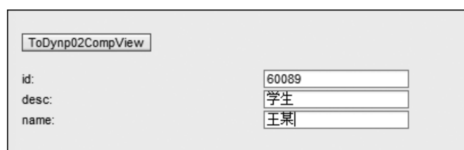


图 3-194　运行效果



图 3-195

15）下面再验证一些其他功能。切换到组件控制器的“Methods”自定义方法中，新建一个 modifyPer( )方法，如图 3-196 所示。

16）输入创建的自定义方法的名称，并选择对应的方法类型，如图 3-197 所示。由于不需要额外设置方法的参数，因此直接单击“Finish”按钮即可。

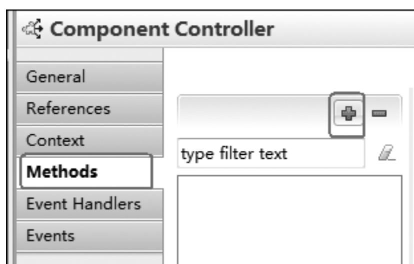


图 3-196

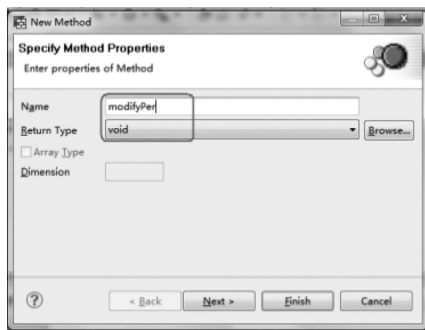


图 3-197　输入方法名称

17）创建方法之后（见图 3-198），保存所有的修改。

18）双击新建的 modifyPer 方法，进入 Java 编辑器页面，如图 3-199 所示。

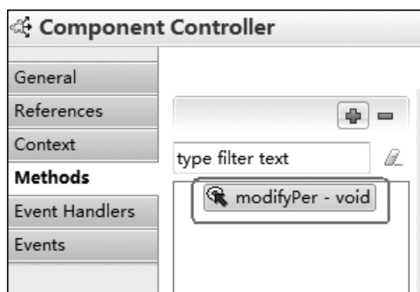


图 3-198　创建方法后的效果

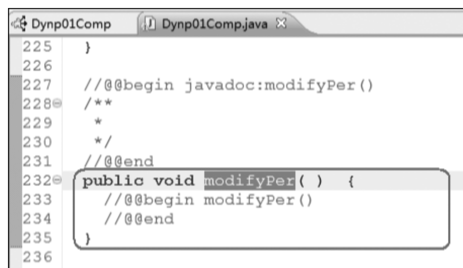


图 3-199　方法的代码编辑

19）在自定义代码域添加以下代码（见图 3-200）：

```
// get currentPerElement Reference
IPerElement currentPerElement = wdContext. nodePer( ). currentPerElement( ) ;
// modify attribute values
currentPerElement. setId( "ID" ) ;
currentPerElement. setDesc( "DESC" ) ;
currentPerElement. setName( "NAME" ) ;
```

```
public void modifyPer( )  {
  //@@begin modifyPer()
      // get currentPerElement Reference
      IPerElement currentPerElement = wdContext.nodePer().c

      // modify attribute values
      currentPerElement.setId("ID");
      currentPerElement.setDesc("DESC");
      currentPerElement.setName("NAME");
  //@@end
}
```

图 3-200　添加代码

20）切换到默认视图页面，在按钮的方法中添加方法的调用（见图 3-201）：

```
wdThis. wdGetDynp01CompController( ). modifyPer( ) ;
```

```
public void onActionToDynp02CompView(com.sap.tc.webdynpro.progm
{
  //@@begin onActionToDynp02CompView(ServerEvent)
    wdThis.wdGetDynp01CompController().modifyPer();
    wdThis.wdFirePlugOut();

  //@@end
}
```

图 3-201　添加代码

21）保存所有修改之后，重新进行编译、部署、运行，运行效果如图 3-202 所示。



图 3-202　运行效果

3.4.5 接口控制器

从整个组件的架构来说，接口控制器（Interface Controller）是整个组件与外部的接口，接口控制器中定义的事件、方法都是可以被外部组件访问的，当然在组件控制器中必须实现接口控制器中声明的这些对象。通常意义上讲，会通过接口控制器来对组件控制器进行规范化，即所有的组件控制器在公共的设计基础之上，遵循了统一的设计原则之后做开发。

开发实例：

1）切换到组件模型编辑器中，单击选中接口控制器，打开属性中的上下文，如图 3-203 所示。

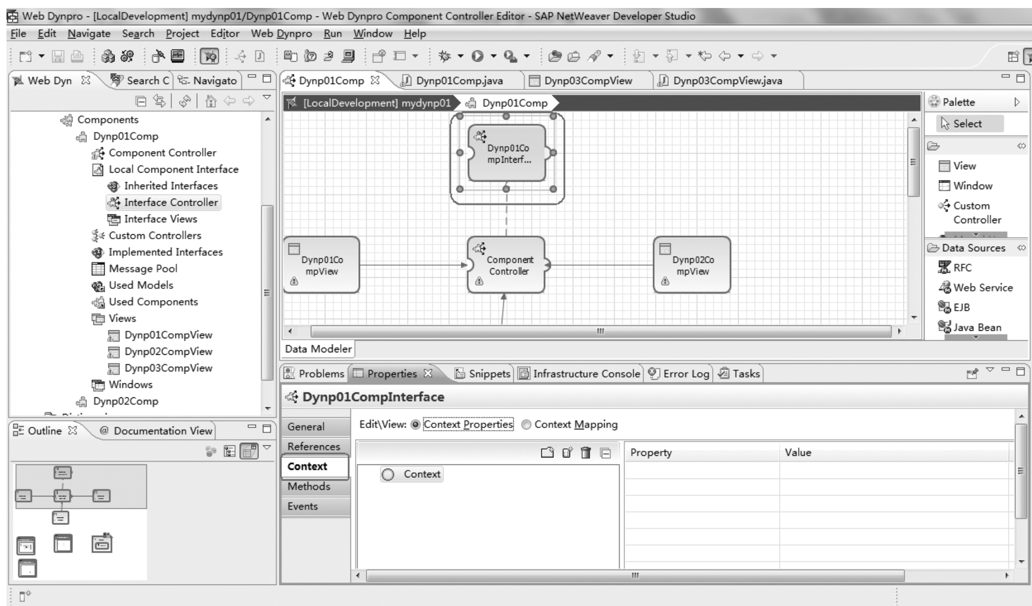


图 3-203 接口控制器的属性

2）添加一组上下文定义。这里添加了集合属性为“1··1”的与流程相关的“ProcessNode”结点，结点中包含“ProcessId”与“ProcessName”两个特性，如图 3-204 所示。

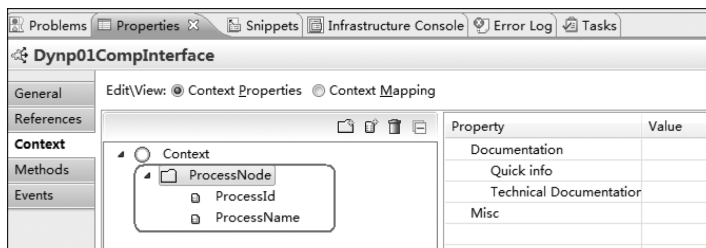


图 3-204 接口控制器中创建上下文

3）保存之后，组件控制器已经报错，报错消息会提示接口中的上下文定义未被实现。复制“ProcessNode”结点至组件控制器的上下文中，如图 3-205 所示。

按照相同的操作方法分别定义方法和事件并在 Java 编辑器中实现。

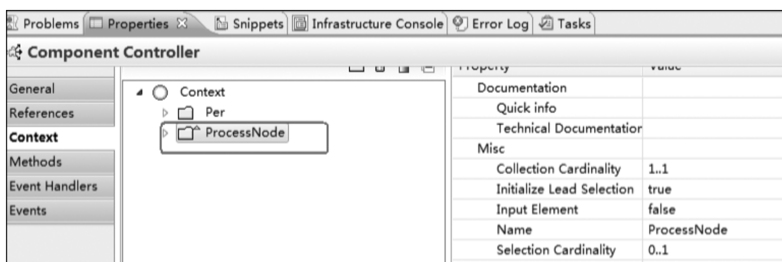


图 3-205　组件控制器中创建上下文

3.4.6　自定义控制器

Web Dynpro 中除了视图控制器、窗口控制器和组件控制器，在 Web Dynpro 工程的目录中还能看到自定义控制器（Custom Controller）。相对而言，视图控制器仅用于界面显示以及一些界面属性预设的控制，与模型的数据交互完全地交给自定义控制器，这样才能算的上是一个好的 MVC 设计。而对于组件控制器来说，自定义控制器可以算是同门兄弟，因为自定义控制器可以实现组件控制器的一些主要功能，每当创建一个自定义控制器，它的设计模式也与上一节中提到的组件控制器的标准类相同；如果将自定义控制器看作通用的控制器，那么组件控制器就是通用控制器中最特殊的一个，如图 3-206 所示。

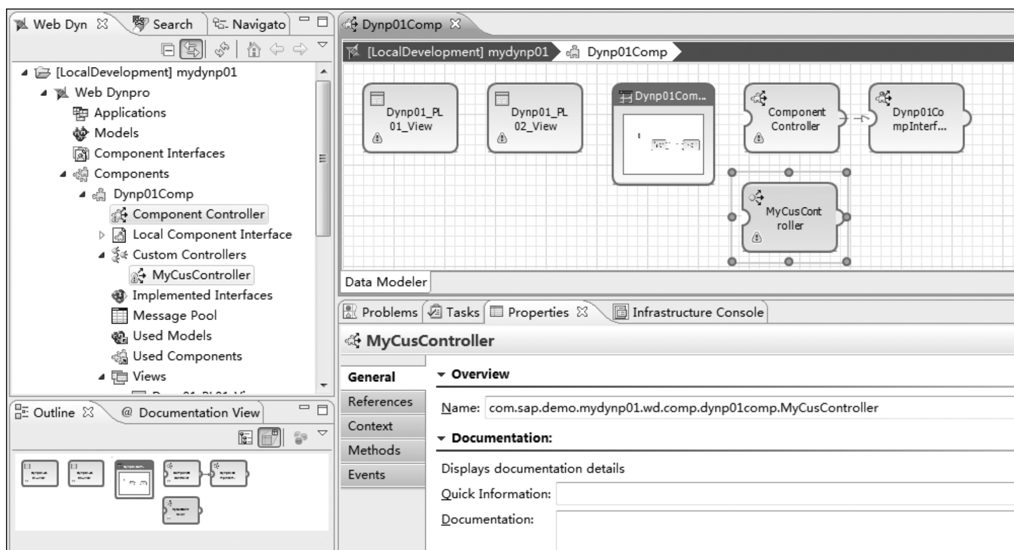


图 3-206　自定义控制器

在实际的项目实施过程中，自定义控制器的目的是用于对 Web Dynpro 模型的处理。在 Web Dynpro 中使用自定义控制器还有以下两个好处：一方面，NWDS 提供了一些模板向导来处理各种各样的模型，同时通过向导能够自动生成自定义控制器对于特定模型的调用方法及具体代码；另一方面，对于 Web Dynpro 中非常复杂的逻辑处理，可以通过不同的自定义控制器进行模块化管理，也有利于后期的代码维护和运维交接。

比较常见的 3 种 Web Dynpro 工程结构如下。

1）视图 – 窗口 – 应用程序：最精简的 Web Dynpro 模型，在视图中定义界面，通过窗口关联相应的应用程序即可运行，也可视图控制器或组件控制器中添加代码进行逻辑处理。

2）模型－自定义控制器－视图－窗口－应用程序：需要处理外部数据源时，整个架构会加入特定的自定义控制器进行逻辑处理。

3）模型－自定义控制器－组件控制器－其他组件－视图－窗口－应用程序：这是最为复杂的 Web Dynpro 程序的模型。

3.5　消息池

Web Dynpro 的消息池可以提供通用的消息定义，方便不同组件和对象使用。双击左侧目录中的“Message Pool”，或者右击“Message Pool”，在弹出的快捷菜单中选择“Open Message Editor”，即可打开消息编辑器视图，对想要显示在屏幕上的消息进行创建和编辑，如图 3-207 所示。

打开消息池后，在右侧的消息编辑器中，开发人员可以维护对应的消息文本、消息类型和消息键。消息文本既可以根据需要翻译成对应的文本，也可以在大量消息内容定义时保持信息的一致性；开发人员在进行源代码开发时，则通过消息键的读取来获取对应的消息文本。最终程序运行时会根据客户端运行环境进行相应的消息显示，如图 3-208 所示。

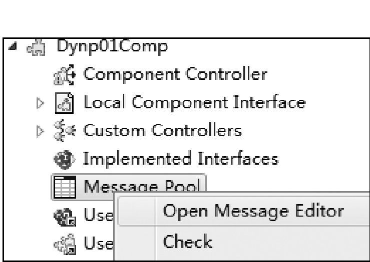


图 3-207　打开消息池

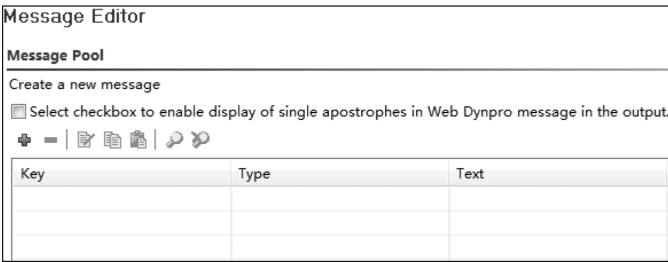


图 3-208　消息编辑器

3.5.1　消息类型

在消息编辑器中，默认允许定义 5 种类型的消息：信息、成功、警告、错误和文本。

前 4 种类型的消息（信息、成功、警告和错误）是 Web Dynpro 中最常用的消息类型，它们都会以提醒消息的方式出现在屏幕上，并带有特定的图标（如成功类型的图标是绿色勾，错误消息类型则是红色叉），而创建这 3 种类型消息都会被作为 IMessage <Component Name> 类中的 Java 常量创建。要注意的是，IWDMessageManager 的实例在同一个组件中始终只有一个。消息应用代码如下：

```
//获取当前组件的消息管理器实例
IWDMessageManager msgMgr = wdComponentAPI.getMessageManager();
//报告对应类型的消息
    msgMgr.reportWarning("This is a ycxWarning!");
```

文本类型的消息并不会作为 IMessage <Component Name> 类中的 Java 常量来创建，它仅用于存放视图中某些 UI 元素的属性值，如标题或文本框等。在控制器代码编写中，文本类型的消息可以通过 IWDTextAccessor 类来获取，当然前提是不要把消息主键弄错。获取消息文本的参考代码如下：

```
//获取当前组件的文本获取实例
IWDTextAccessor txtAcc = wdComponentAPI. getTextAccessor( );
//通过消息键获取消息池中的文本信息
String myTextMsg = txtAcc. getText(" <MessageKey> ");
```

3.5.2 消息区域

默认情况下，当提醒消息被定义时，它会显示在页面的下方。接触过视图 UI 元素的开发人员可以进一步发现，其实消息会以 Web Dynpro 中透明表格的方式出现在视图的最下方。如果视图中的元素太多，或者页面过长，则放在最下方的消息很容易被用户所忽略。所以 Web Dynpro 提供了自定义消息区域位置的 UI 元素 "Message Area"，开发人员可以在设计视图时根据实际需要对消息区域的位置进行个性化设置，如图 3-209 所示。

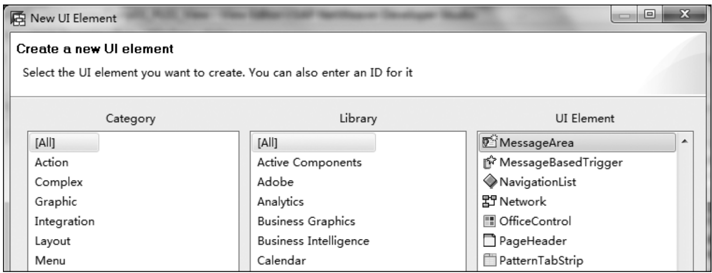


图 3-209　消息区域的视图元素

3.5.3 多语言配置（I18N）

SAP 通过不同版本的语言配置文件来实现多语言功能，同时，考虑到后期维护的工作量，SAP 也不建议开发人员在一个国际化的程序的代码中编写指定语言的消息代码。在一个 Web Dynpro 工程中，多语言版本的配置不需要进行额外的代码编写，普通业务人员或最终用户都可以完成，这个过程在 SAP 的一些文档中被称为 "Internationalization"。多语言配置的优点也是显而易见的：不再需要为了不同语言版本而准备不同版本的代码。

注：在一些文档中 "Internationalization" 被简化为 "I18N"，意思是一个以 "I" 开头、"N" 结尾再加上中间 18 个字母的单词。

在 Web Dynpro 中，多语言的配置都是存放在扩展名为 .xlf 的描述文件中。Web Dynpro 的元素与描述文件的对应关系见表 3-6。

表 3-6　Web Dynpro 的元素与描述文件的对应关系

Web Dynpro 的元素	对应的 . xlf 描述文件命名
视图（View）	[NAME]. wdview. xlf；[NAME]. wdcontroller. xlf
窗口（Window）	[NAME]. wdwindow. xlf
消息池（MessagePool）	[NAME]MessagePool. wdmessagepool. xlf
自定义类型（SimpleType）	[NAME]. dtsimpletype. xlf

多语言配置暂时不能通过类似创建开发工程的向导进行配置操作，具体过程如下：

1）通过开发工具中的 Navigator 或者文件目录的查找功能，定位对应的 . xlf 描述文件。

参考路径为[PROJECT]/src/package/[PACKAGE]…/[PROJECT NAME]/wd/comp/ * . xlf。

2）将对应的 . xlf 文件复制、粘贴在同一目录下，并根据目标语言的 ISO 编码重命名：

如需翻译为德语，则将 example. xlf 复制、粘贴在同一目录后，重命名为 example_de. xlf。

部分常用语言编码见表 3-7。

表 3-7　部分常用语言编码

语　　言	ISO 编码	语　　言	ISO 编码
英语	en	意大利语	it
德语	de	俄语	ru
法语	fr	日语	jp

3）双击从 NWDS 开发工具中打开重命名后的 . xlf 描述文件，即可对具体的文本内容进行翻译配置。

4）在编译之后新的 . xlf 描述文件会对应生成一个以 . properties 为扩展名的配置文件，供程序在运行时调用。

以 example_de. xlf 为例，在编译后会生成一个 example_de. properties 的文件。

需要注意的是，多语言配置并不能覆盖 Web Dynpro 工程中的所有文字，有一部分内容是在开发的同时已指定相应语言，且无法通过多语言配置实现翻译内容的展现的，具体包括以下内容：

1）UI 界面元素中的输入文本和提示消息文本。

2）视图控制器中方法、动作的描述文本。

3）窗口（Window）的标题。

4）Java 字典中的文本内容和描述。

开发实例：

1）新建 Web Dynpro 工程 Dynp09，创建一个组件和视图 Dynp09_01Comp_01View，任意创建一些 UI 元素，视图效果如图 3-210 所示。

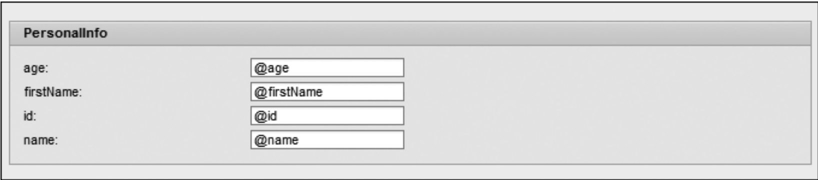


图 3-210　视图效果

2）通过 Project Navigator 查找该 View 对应的 . xlf 描述文件 Dynp09_01Comp_01View. wdview. xlf，如图 3-211 所示。

3）复制该描述文件并粘贴至同一目录下，重命名为 Dynp09_01Comp_01View. wdview_de. xlf，如图 3-212 所示。

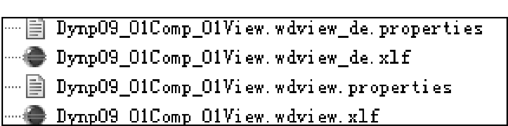


图 3-211　定位 . xlf 文件

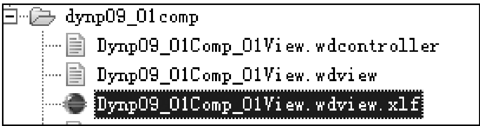


图 3-212　创建新描述文件

注：截图中的 properties 文件是编译后生成的。

4）双击新命名的德语文件，从 NWDS 的默认编辑器打开，单击下方的“Resource Text”页签切换至页面元素和文本内容，如图 3-213 所示。

[LocalDevelopment] mydynp09 | Dynp09_01Comp | Dynp09_01Comp_01View | Dynp09_01Comp_01View_ch.wdview.xlf

Resource name	Resource Type	Text	Max Length	Translate
RootUIElementContai...	label	age	255	yes
RootUIElementContai...	label	firstName	255	yes
RootUIElementContai...	label	id	255	yes
RootUIElementContai...	label	name	255	yes
RootUIElementContai...	caption	PersonalInfo	255	yes

Add
Edit
Delete
Set Filter
Activate Filter

图 3-213　编辑描述文件

5）选中“lable”行，单击“Edit”按钮，即可在对应文本框中输入相关德语描述，如图 3-214 所示。

Resource Type: label
Resource name: RootUIElementContainer/Child:Group/Child:TransContainer_0/Child:age_label@text
Text: Alter
Maximum Length: 255
Translate

图 3-214　修改对应文本

6）用同样的方法翻译所有的描述并保存，如图 3-215 所示。

Resource name	Resource Type	Text	Max Length	Translate
RootUIElementCo...	label	Alter	255	yes
RootUIElementCo...	label	Vorname	255	yes
RootUIElementCo...	label	Identifikation	255	yes
RootUIElementCo...	label	Name	255	yes
RootUIElementCo...	caption	PersonalInfo	255	yes

图 3-215　修改后的效果

7）保存所有修改，重新编译、运行之后，即可进行相关测试，运行效果如图 3-216 所示。

PersonalInfo
age:
firstName:
id:
name:

图 3-216　运行效果

8）浏览器语言设置为德语，如图 3-217 所示。



图 3-217　设置浏览器语言

9）刷新应用，页面自动成为德语版，如图 3-218 所示。

PersonalInfo

Alter:

Vorname:

Identifikation:

Name:

图 3-218　运行效果

3.6　组件

介绍过了 Web Dynpro 中的三大件（视图、窗口、控制器），下面重新介绍一下 Web Dynpro 的组件（Component）。

3.6.1　组件的生命周期

1. 运行生命周期

由于组件直接对应 Web Dynpro 的应用程序，并通过浏览器 URL 进行访问，因此从前台访问请求开始到对应服务器整个会话实例销毁，即称为组件的一个生命周期。组件控制器作为组件的核心控制组件，其生命周期伴随着整个组件的始终，而不同类型的窗口视图具有不同的生命周期，但其生命周期都包含于组件的生命周期。下面介绍组件中各个对象的生命周期。

1）组件控制器：组件控制器依赖于整个组件应用，可以说组件控制器的生命周期就是整个组件应用的生命周期。

2）窗口：分为两种情况，第一种是实现了接口视图的窗口，当与接口视图相关的应用程序被访问时，窗口控制器的生命周期开启，此时与组件控制器生命周期一致，并依赖于整个组件应用周期（会话的访问周期）；第二种是没有实现接口视图的窗口（外部窗口或确认

窗口)，对于没有实现接口视图的窗口，则是通过窗口处理的 API 去创建或销毁窗口实例，这种类型窗口的生命周期即是从实例创建到销毁的时期。

3）视图：视图的生命周期较为短暂，同一窗口中视图的切换就可以结束一个视图实例，视图也不依赖组件中其他对象组件，但在组件的生命周期中，视图控制器通常采用与组件控制器上下文映射来保持数据的持久性。

2. 逻辑生命周期

组件在不同的生命周期时间点会执行相应的操作，以控制器中的一些标准方法为例，wdDoInit 方法是一个组件生命周期的开始（初始化操作)，wdDoExit 为一个组件生命周期的结束操作（程序的出口)。下面针对几种控制器相对整个组件应用来分析它们的初始化执行顺序。

1）组件控制器：组件应用初始化时，组件控制器执行 wdDoInit( )方法，在整个组件的生命周期中，它只执行一次并且最先执行。

2）窗口控制器：当一个 Web Dynpro 应用程序被访问时，实现了默认接口视图的窗口就会被初始化，并执行组件控制器当中的 wdDoInit( )方法。窗口控制器的初始化动作在组件控制器之后执行。由于此时窗口控制器的生命周期与组件一致，因此窗口控制器的初始化也只执行一次。而没有实现接口视图的窗口（一般在代码中实现)，它的初始化方法会随着整个窗口实例的创建与销毁反复执行。

3）视图控制器：视图控制器的生命周期在整个组件应用生命周期当中，它的初始化也会随着视图实例的创建与销毁反复执行。

需要强调的是，在不同场景中，可能需要创建多个控制器的实例，所以不同的控制器可能在整个组件应用实例的生命周期中执行多次，如弹出窗口、视图跳转等都是控制器生命周期的体现。在这个操作过程当中，窗口或视图实例都会被反复创建、销毁，窗口或视图的初始化也会执行多次，但对于同一个组件实例来说，它的生命周期只有一次，在它的生命周期当中，初始化操作也只执行一次。

3.6.2 组件之间的访问

一个 Web Dynpro 组件既是一个独立的开发单元，从整个项目来看也是一个可被复用单元模块，所以每个组件都可以成为别的组件的引用或是引用别的组件。如果能合理有效地对整个开发项目进行规划和架构，则会带来以下几点优势：

1）多组件的架构反而能使得整个项目更易于管理和后期维护。

2）针对基础公共功能进行统一管理和开发，节约开发时间和人力成本。

3）在多组件组成的大型工程中，父组件不需要了解子组件的数据结构、处理逻辑等内部信息，只需要关注子组件的接口即可，这与大家更熟悉的“面向对象”开发有着异曲同工的优势。

1. 内部访问

首先要说明的是，“内部访问”并不是说组件内部访问，而是只涉及视图、窗口和组件控制器之间的引用。这里“内部访问”是指 Web Dynpro 工程内部多个组件之间的访问。下面通过一个开发实例来体验一下。

开发实例：

1）在 Web Dynpro 的子目录中，右击“Components”组件，在弹出的快捷菜单中选择“Create Component”，如图 3-219 所示。

2）在弹出的对话框中输入组件的名称“Dynp03Comp”，并选择存放的目录，下方默认已选中创建默认的视图和窗口，如图 3-220 所示。

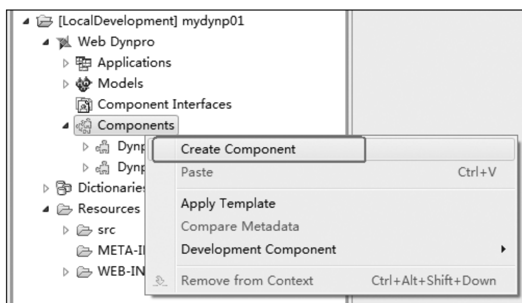


图 3-219　创建组件



图 3-220　输入组件名称

3）由于选中了创建自定义的视图和窗口，因此在弹出的窗口需填写自定义窗口的名称以及对应的接口视图名称（见图 3-221），完成后单击“Finish”按钮。

4）完成后记得报错所作的修改，右击新建的视图，在弹出的对话框中选择“Refactor”→“Rename”，对视图 Dynp03Comp01View 进行重命名，如图 3-222 所示。

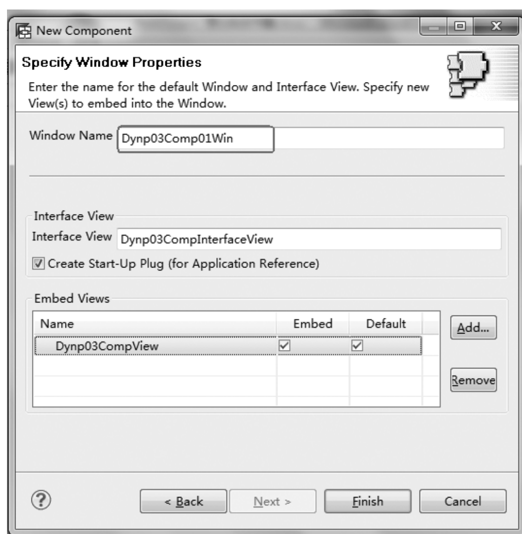


图 3-221　输入窗口和视图名称

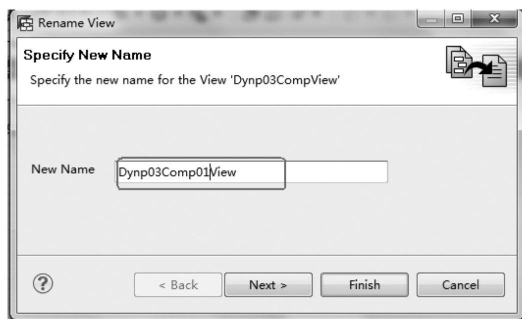


图 3-222　输入新名称

5）以之前创建的“modifyPer”方法为例，将它公开给新创建的组件进行调用。不论是方法还是上下文，如果需要被其他组件“发现”，就必须通过接口控制器进行公开声明。

切换到已经完成配置的组件 Dynp01Comp，复制组件控制器的自定义方法 modifyPer( )到接口控制器中，如图 3-223 所示。

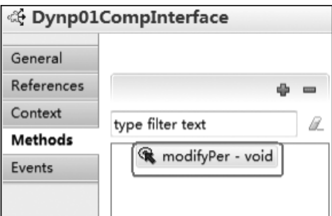


图 3-223　创建自定义方法

6）回到 Dynp01Comp 的组件控制器中，修改 modifyPer( )实现，添加消息打印代码（见图 3-224）：

```
wdComponentAPI. getMessageManager( ). reportSuccess( currentPerElement. getId( ) );
```

```
public void modifyPer( )  {
  //@@begin modifyPer()
      // get currentPerElement Reference
      IPerElement currentPerElement = wdContext.nodePer().cur

      // modify attribute values
      currentPerElement.setId("ID");
      currentPerElement.setDesc("DESC");
      currentPerElement.setName("NAME");
      wdComponentAPI.getMessageManager().reportSuccess(
              currentPerElement.getId());
  //@@end
}
```

图 3-224　添加代码

7）至此完成了代码的准备工作，下一步在新创建的 Dynp03Comp 组件中添加引用。一种方式是双击打开组件控制器，在属性的“Reference”引用子标签中添加子组件；另一种方式是在“Used Components”使用的组件中直接添加。

展开 Dynp03Comp，右击“Used Components”，在弹出的快捷菜单中选择“Add Used Component”，如图 3-225 所示。

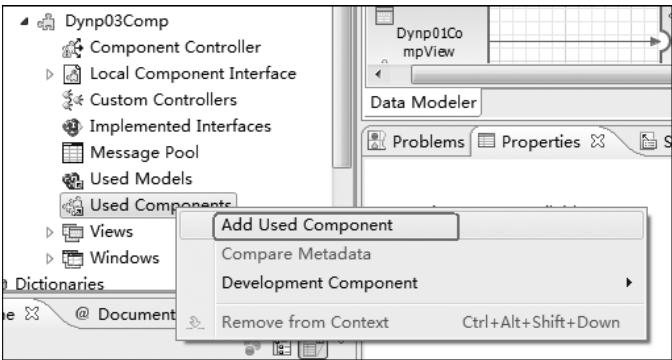


图 3-225　添加组件引用

8）在弹出的窗口中单击浏览按钮，选择“Dynp01Comp”，如图 3-226 所示。

9）输入引用组件的名称“Dynp01Comp”，这里可以理解为在计算机操作系统中创建快

捷方式时定义的快捷方式名称，单击“Finish”按钮，如图 3-227 所示。

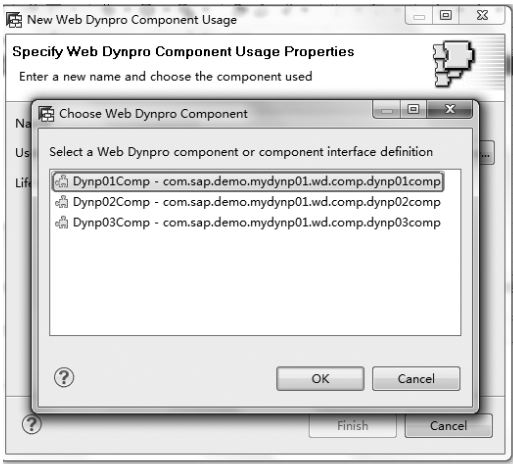


图 3-226　选择组件

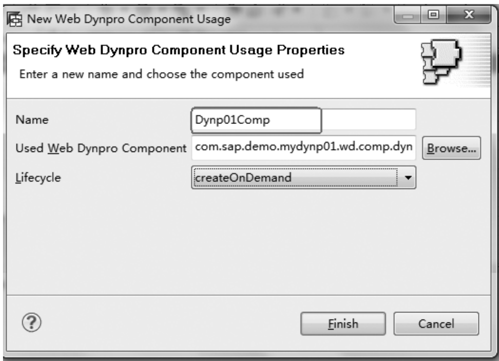


图 3-227　输入引用组件名称

10）完成后保存所有修改，此时在“Dynp03Comp”组件（以下简称 03 组件）的模型编辑器里已经能看到成功添加了原“Dynp01Comp”组件（以下简称 01 组件）的接口控制器，如图 3-228 所示。

11）单击“Data Link”按钮（见图 3-229），单击 03 组件的组件控制器并拖曳至 01 组件的接口控制器，完成对其的引用。

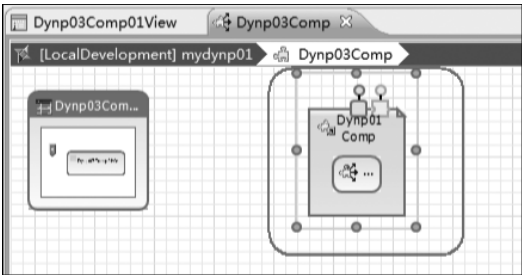


图 3-228　添加后的效果

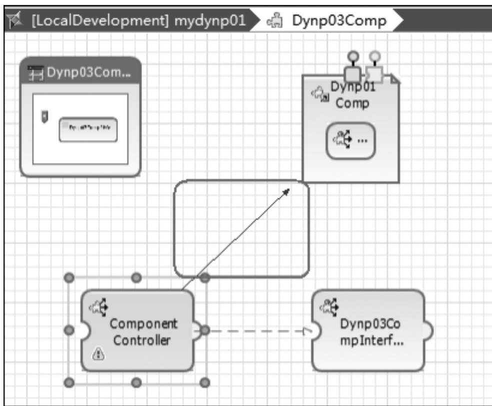


图 3-229　添加数据链接

12）在弹出的窗口中可以进行组件之间的上下文映射，左侧是 03 父组件，右侧是被引用的 01 子组件，如图 3-230 所示。由于本实例不需要进行映射配置，因此直接单击“Finish”按钮即可。

13）在 03 组件的组件控制器定位到 wdDoInit( ) 方法并添加 01 组件的自定义方法，添加以下代码（见图 3-231）：

```
wdThis. wdGetDynp01CompInterface( ). modifyPer( ) ;
```

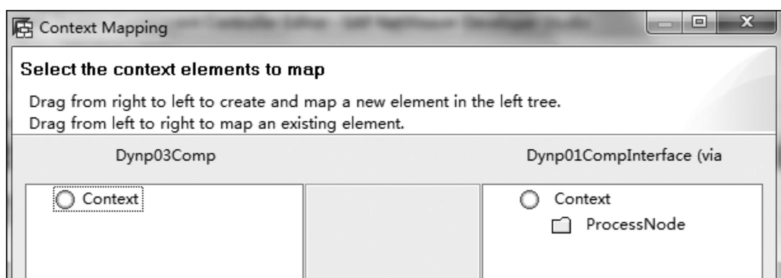
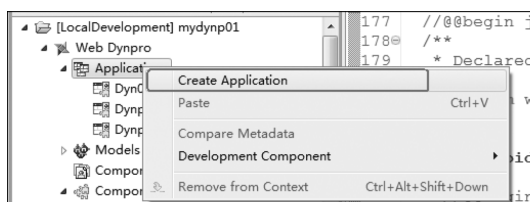


图 3-230　编辑上下文映射



```
public void wdDoInit()
{
  //@@begin wdDoInit()
    wdThis.wdGetDynp01CompInterface().modifyPer();
  //@@end
}
```

图 3-231　添加代码

14）创建一个以 03 组件为默认界面的应用程序。右击“Application”，在弹出的快捷菜单中选择“Create Application”如图 3-232 所示。

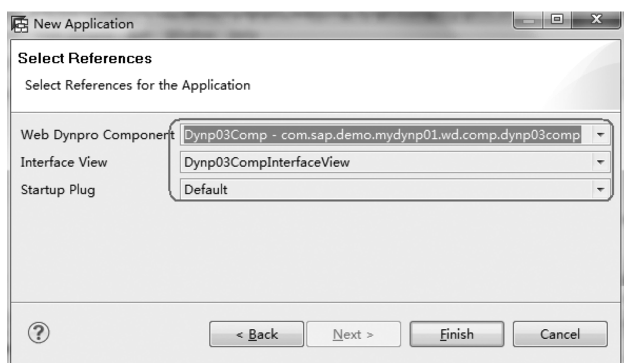


图 3-232　创建应用

15）输入应用程序名称，选择 03 组件及其接口视图，如图 3-233 所示。

图 3-233　选择接口视图

16）保存所有修改后，编译、运行该应用程序，即可看到 03 组件调用 01 组件方法成功，如图 3-234 所示。

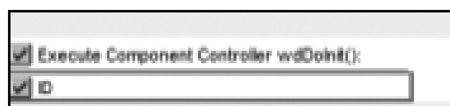


图 3-234　运行效果

2. 外部访问

从 Web Dynpro 工程的外部来看，或者从 SAP 的 Java 服务器的角度来看，每一个工程其实都是一个开发组件（Development Component，DC），SAP 建议 SAP 的开发实施工程都以 DC 的形式进行，并存放在对应的开发架构中。对于默认的 Web Dynpro 工程，其实之前的开发实例都是存放在了一个本地的开发组件中，如图 3-235 所示。

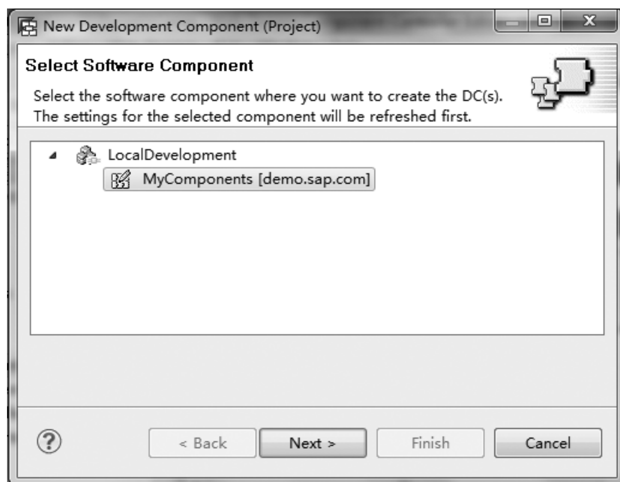


图 3-235　本地开发组件

如果需要进行多个开发人员的协同开发，则所有相关的开发工程都需要上传至服务器端的开发架构进行版本同步和管理，并且 SAP 的开发组件支持许多不同类型的开发工程，如图 3-236 所示。

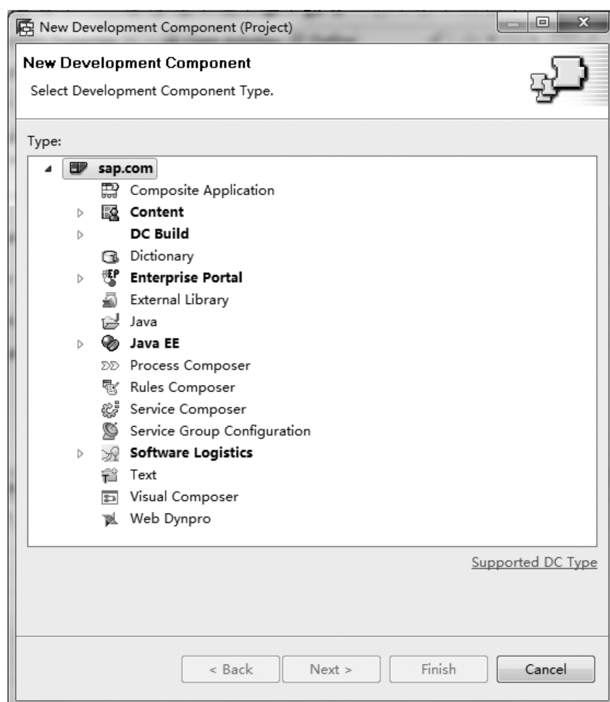


图 3-236　开发组件的不同类型

不论是什么类型的开发组件，都包含了所有类型的开发对象（类、接口及表结构等内容），同时它有一个统一的对外接口，叫作开发对象的“Public Part”（公共部分，PP）。以之前的 Web Dynpro 工程为例，在需要编译时，除了右击工程名出现的“Rebuild”，还可以选择“Development Component”→“Build”进行编译。如果工程涉及外部访问，则必须使用第二种操作进行编译，即必须以开发组件进行编译，这个编译过程会将每个公共部分编译成为一个独立的 JAR 文件。因此如果不是通过开发组件的编译进行操作，则公共部分的内容就无法更新，工程也就无法被外部访问。

在项目的架构和设计中，经常会抽象出诸多基础功能，这些基础功能会根据涉及内容不同而最终落脚为几个基础功能开发组件，其他开发组件通过对基础功能 DC 的依赖、引用，去调用基础部分的功能，如审批意见、附件管理、日志记录或数据库连接等。

开发实例：

1）在本实例中，假设 mydynp01 工程为基础功能，则需要新建另一个 Web Dynpro 工程对其进行访问和调用。在空白处右击或单击开发工具中左上角。“新建”按钮，选择新建 Web Dynpro 开发组件，如图 3-237 所示。

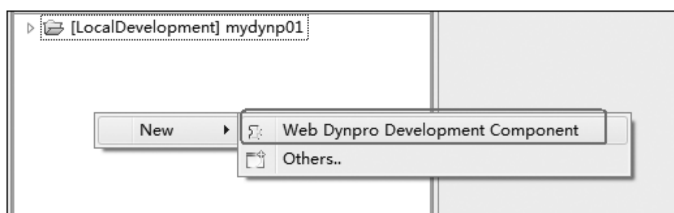


图 3-237　创建 Web Dynpro 开发组件

2）选择本地开发（后续章节会有服务端同步开发实例），单击“Next”按钮，如图 3-238 所示。

3）输入自定义 Web Dynpro 开发组件名称和其他必要信息（见图 3-239），单击“Next”按钮。

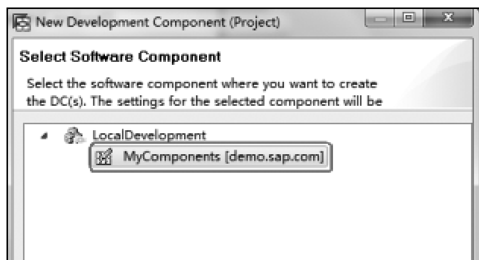


图 3-238　选择本地组件

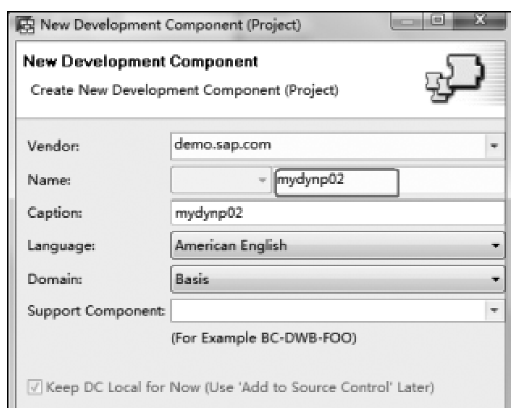


图 3-239　输入项目信息

4）完成工程创建后，右击“Components”组件，在弹出的快捷菜单中选择“Create Component”如图 3-240 所示。

5）按照之前的步骤，创建“Dynp02_01Comp”组件，并默认创建“Dynp02_01Comp_

01Win”窗口和“Dynp02_01Comp_01View”视图，如图 3-241 所示。

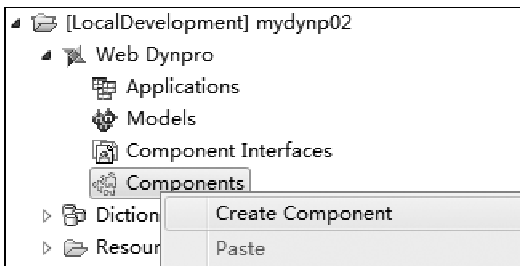


图 3-240 创建组件

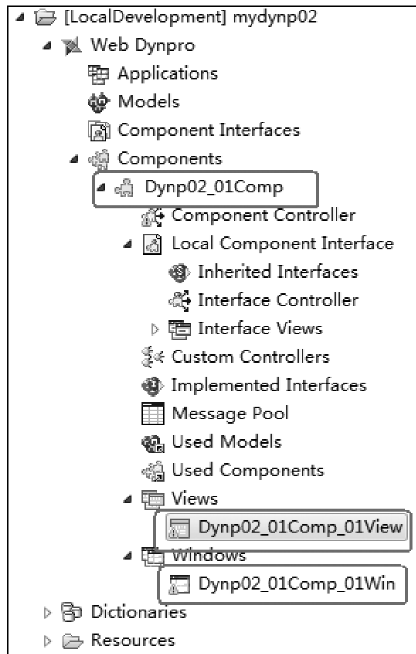


图 3-241 创建后的效果

6）通过切换按钮，切换到“Development Infrastructure”，如图 3-242 所示。

7）在左侧列表中找到“MyCOmponents”即可看到我们创建的 Web Dynpro 项目，如图 3-243 所示。

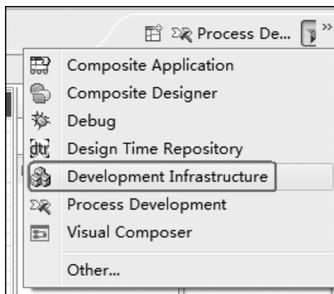


图 3-242 切换透视图



图 3-243 项目位置

注：工程图标左上角的雪花表示最新的修改还未以开发组件形式编译，编译后雪花标识就会消失。

8）定义 mydynp01 的 PP，选中 01 工程，单击公共部分页签，单击“Add”按钮，如图 3-244 所示。

9）在弹出的窗口中输入自定义公共部分名称，单击“Finish”按钮，如图 3-245 所示。

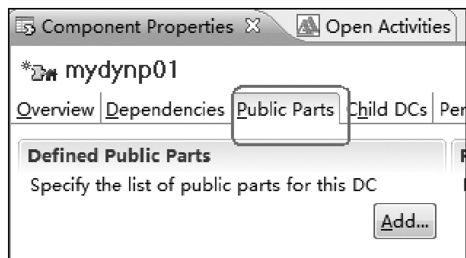


图 3-244 公共部分属性

10）将公开的部分添加至公共部分，右击“api”，在弹出的快捷菜单中选择“Manage Entities”，如图 3-246 所示。

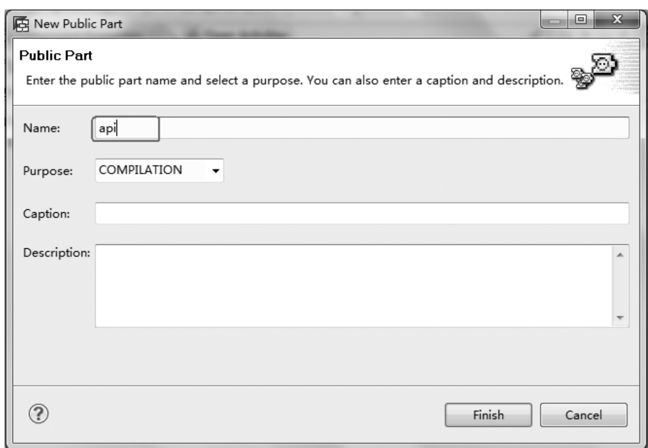


图 3-245　输入公共部分名称

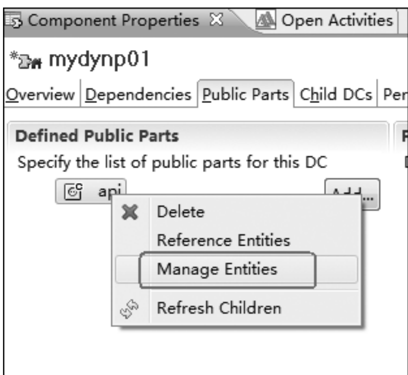


图 3-246　管理实体的菜单选择

11）在弹出的窗口中选择 Dynp01Comp，单击“Finish”按钮，完成公共部分配置，如图 3-247 所示。

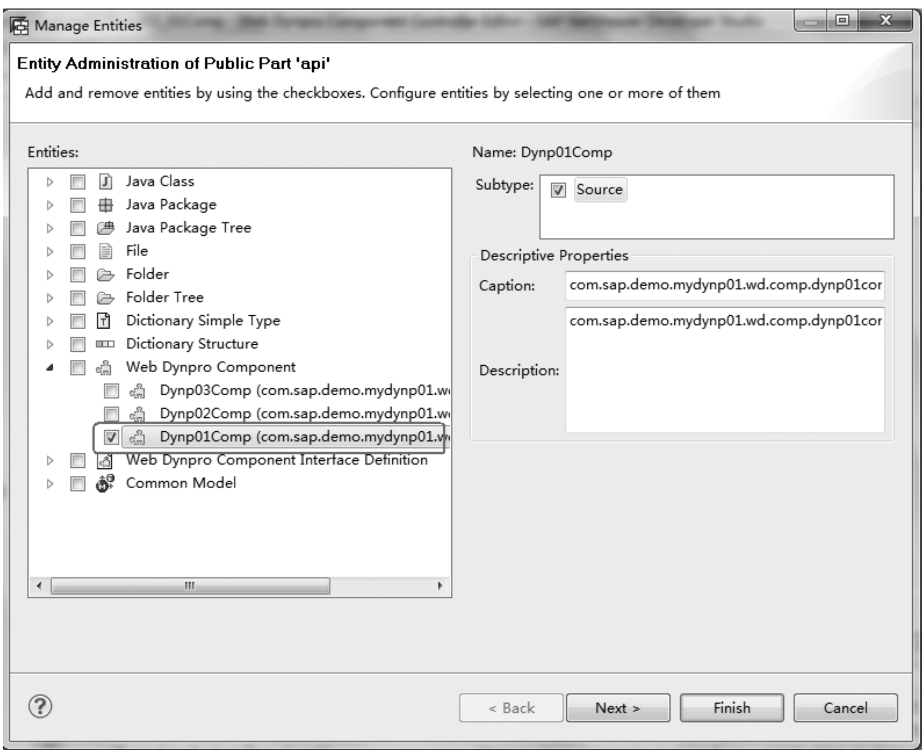


图 3-247　选择添加内容

12）保存修改后，新建的公共部分中已包含 Dynp01Comp 组件，如图 3-248 所示。

13）接下来添加新建的开发组件 mydynp02 对 mydynp01 的依赖，如图 3-249 所示。选中 mydynp02 后，打开“Dependencies”依赖页签，单击“Add”按钮。

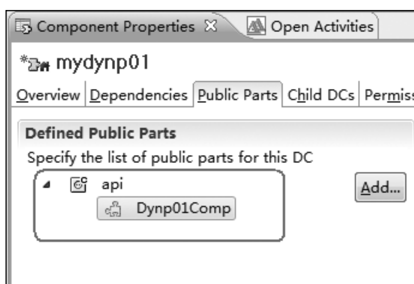


图 3-248　添加后的效果

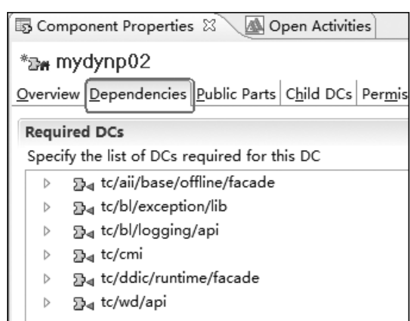


图 3-249　依赖属性

14）在弹出的窗口中找到 mydynp01 并选中，单击“Next”按钮，如图 3-250 所示。

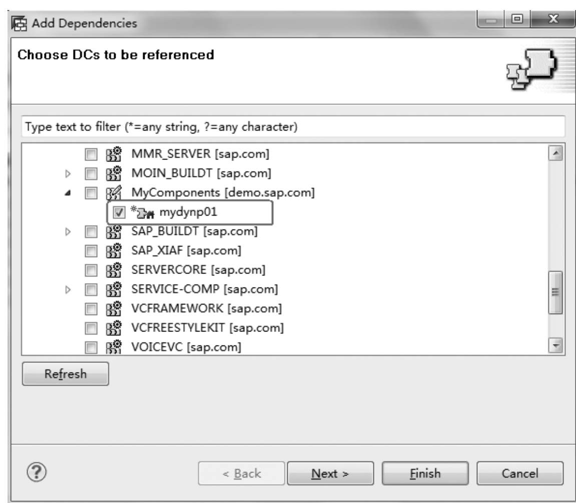


图 3-250　选择对应开发组件

15）根据实施需求对依赖明细进行配置，单击“Finish”按钮，如图 3-251 所示。

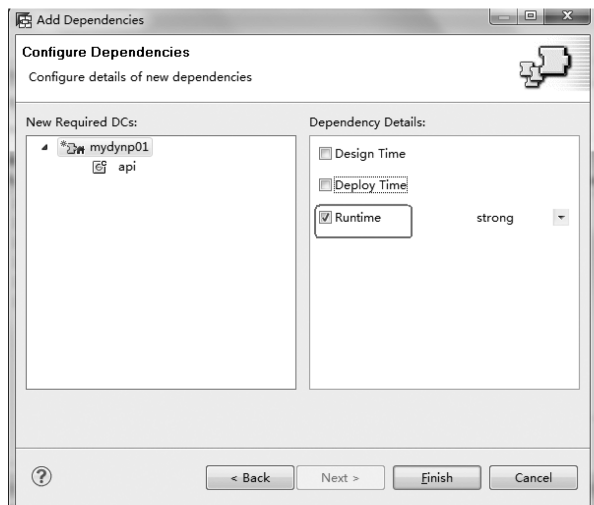


图 3-251　选择依赖方式

至此已完成所有开发组件的配置，记得保存所做修改并以开发组件方式编译。

16）切换回到 Web Dynpro 透视图，在“mydynp02”中添加对“mydynp01”组件的引用（见图 3-252），右击“Used Components”，在弹出的快捷菜单中选择“Add Used Component”，在弹出的浏览窗口中新增了“Dynp01Comp”组件，即为新添加依赖“mydynp01”的公共部分“api”内容，选中该组件即可。

17）双击“Dynp02”项目的组件控制器，进入模型编辑器，通过数据链接的连线，创建组件控制器到“Dynp01Comp”接口控制器的关联，如图 3-253 所示。

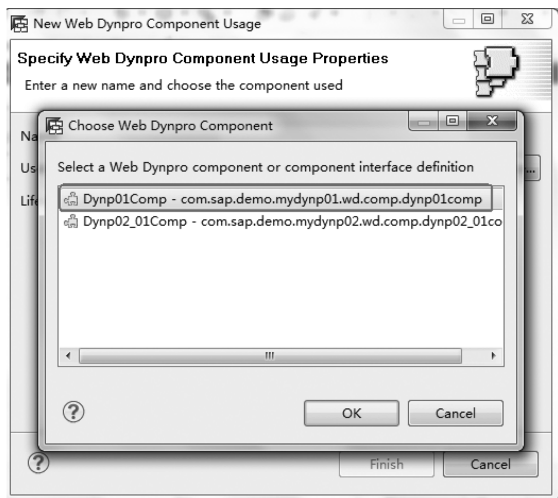


图 3-252　添加组件引用

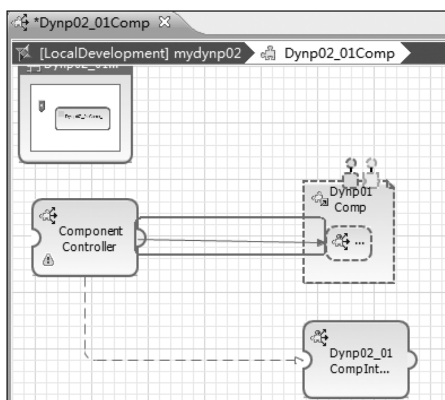


图 3-253　添加后的效果

18）在“Dynp02”组件的组件控制器定位到“wdDoInit( )”方法，并添加以下代码（见图 3-254）：

```
wdThis. wdGetDynp01CompInterface( ). modifyPer( );
```

```
public void wdDoInit()
{
  //@@begin wdDoInit()
    wdThis.wdGetDynp01CompInterface().modifyPer();
  //@@end
}
```

图 3-254　添加代码

19）创建一个以“Dynp02”组件为默认界面的应用程序，并部署运行，运行效果如图 3-255 所示。



图 3-255　运行效果

3.7　模型

Web Dynpro 模型为应用程序可以访问不同类型的数据源（如 RFC、WebService 或 EJB）

提供了标准的接口，换句话说，Web Dynpro 通过创建模型的形式自动生成了规范化的客户端代理类，开发人员只需要对代理类做操作便可以快速完成对接口的调用。

3.7.1 模型与自定义控制器

1. 自定义控制器在使用模型过程中起到的作用

组件控制器其实是最特殊的一个自定义控制器，而一般的自定义控制器并不作为全局的控制与中转器，而是更多关系到后台服务，开发工具提供模型导入的向导并将模型结构转化为本地可识别的代理类，自定义控制器可以进一步地将模型的数据结构定义到上下文中，同时初始化接口结构数据并生成了调用接口的方法。换言之，通过配置生成的自定义控制器中，它的上下文包含了接口的输入/输出参数，它的控制器（类文件）中包含调用接口的方法，对上下文输入结构赋值，执行接口方法，最后通过上下文中的输出结构获取接口返回数据。

2. 规范化地管理自定义控制器

事实上，自定义控制器与模型可以是多对多的关系，一个自定义控制器可以通过应用模板的方式（Apply Template）引用多个模型，同时一个模型根据接口方法的不同又可以创建多个自定义控制器，理论上这几种映射关联的方式都是可行的，但从开发设计的角度来说，尤其是考虑到良好的重用性与后期的维护，这样做并不符合规范。严格地来讲，一个自定义控制器对应一个模型，当模型中含有较多的接口方法时，比如 WebService 中有若干执行方法，那么自定义控制器中就含有多个接口调用方法；当模型中只含有一个方法时，自定义控制器就对应只有一个接口方法以及方法对应的输入/输出结构，如图 3-256 所示。这样做，如果接口结构变，不会影响到其他接口，同时项目开发或后期维护都比较清晰，便于修改。

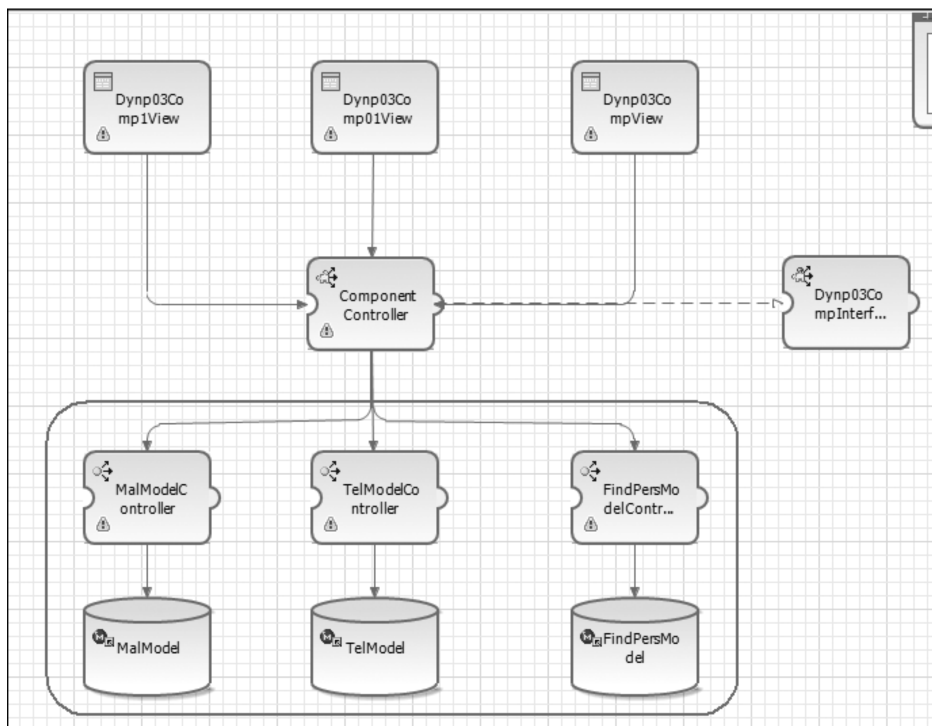


图 3-256　自定义控制器与模型

3. 模型与组件控制器的关系

组件控制器在组件中起控制与中转的作用，视图与视图之间、视图与其他控制器之间的数据传递都是通过组件控制器进行，同样前台的 UI 在访问后台服务时也是通过组件控制器中的上下文进行传递。

控制器与控制器之间的数据传递与调用并不是都要经过组件控制器，实际上 Web Dynpro 在设计时并没有做这方面的硬性限制，但从系统设计的角度，以及从代码的重用与可维护性上来讲，需要遵循这样一种模式，即在本章最开始介绍的 MVC 模式。当把所有的应用逻辑都封装在组件控制器中，把基本的数据定义在上下文结构中时，就能体会到代码维护一致性的便利。

一般情况下，开发人员都会遵循这样一个流程：视图首先调用组件控制器中的方法，组件控制器引用自定义控制器并调用自定义控制器中与模型映射产生的方法，最后自定义控制器通过调用系统自动生成的模型客户端代理类来调用后台接口，同样视图、组件控制器和自定义控制器之间都可以通过上下文映射的方式实现数据的共享。在共享过程中，自定义控制器作为源结点，视图通过上下文中的输入参数为接口赋值，通过上下文中的输出参数获取接口的返回数据，如图 3-257 所示。

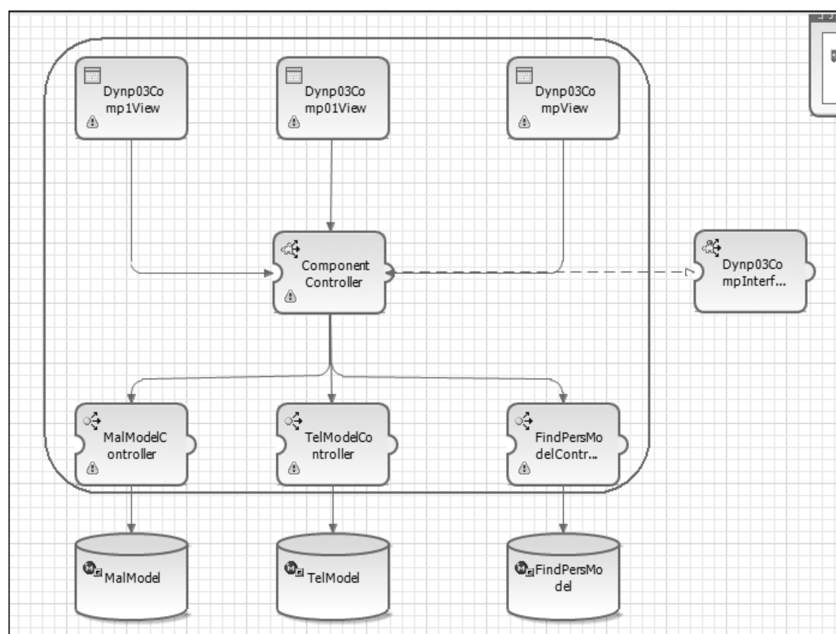


图 3-257　自定义控制器与组件控制器

3. 7. 2　RFC 模型

RFC（Remote Function Call，远程功能调用）是 SAP 为了提供其他系统可以调用 ABAP 功能而提供的一种技术，是 SAP ERP 与其他系统（包括 SAP 系统或非 SAP 系统）间的一种双向接口技术，也是 SAP 与外部通信的基本协议。RFC 是基于是 CPI－C（Common Programming Interface for Communications layer）实现的，在最初 RFC 创建时，只能被 C 语言的程序调用，后来为了适应网络时代的变化，也允许 Java 语言进行访问，而在 Web Dynpro 中 RFC 自然也允许其通过企业连接器进行访问。

简单来说 RFC 的作用是系统调用当前系统外的程序模块，从而实现某个功能，但两个系统中至少有一个必须是基于 ABAP 应用服务器的系统。当然 RFC 也可以在 ERP 系统内被本系统的程序调用，而通常情况下都是为跨系统调用而使用的。

RFC 的功能包含以下几点：

- 登录并退出远程 ABAP 系统（在 Web Dynpro 配置中，导入模型使用的 ABAP 系统配置不必与实际运行时的 ABAP 系统保持一致），并进行权限检查。
- 调用并管理远程系统会话所需的通信线程。
- 自动将功能模块中的参数转换成远程系统所需的格式，或反之将远程系统需要的格式转换为 SAP 中的参数。其中可能包含不同技术平台之间的转换。
- 处理通信过程中的错误，如果需要，也可返回报错明细给调用方。
- 可以对外声明，负责远程调试的实现。

在 SAP ABAP 系统中（包括 R/3 或 ECC、BW、PI 等系统），登录用户都可以在权限允许的前提下通过 SE37 事务代码进行功能模块的创建和修改，而只有在属性中将功能模块的类型选择为“Remote - enabled module”才能被外部系统调用，有时非 ABAP 开发人员会一直无法搜索到对应的 RFC 名称，很有可能就是因为没有修改它的属性导致的，如图 3-258 所示。

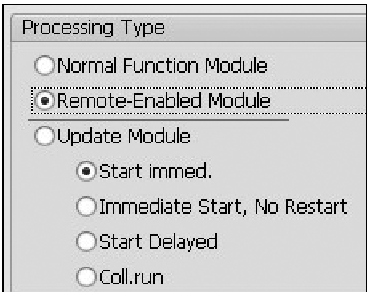


图 3-258　RFC 的关键属性

以下以测试系统中的 ZTEF_001 功能模块为例进行简单介绍，登录 ECC 系统后，输入事务代码“SE37”，随后输入函数名称并单击“Display”按钮，如图 3-259 所示。



图 3-259　RFC 中定义的字段

每个功能模块都包含了以下几个页签，见表 3-8。

表 3-8　功能模块中的页签说明

页签名称	使用说明
Attributes	属性页，配置功能模块的相关属性
Import	输入参数，类似 Java 方法中的输入参数
Export	返回参数，类似 Java 方法中的返回参数
Changing	输入/输出都需要使用的参数
Tables	功能模块用到的表和结构
Exceptions	功能模块抛出的异常参数
Source code	功能模块的源代码

如果使用过老版本的 NWDS，也能看到类似 ECC 中功能模块配置页的“影子”，其中“Properties”对应着属性页签，“Context”对应着参数页签，“Implementation”对应着源代码页签，而在新版本中，上下文、方法、事件等内容都统一放入了“Properties”属性页签中，Java 源代码也独立成为一个 Java 文件，并可以通过 Java 编辑器进行修改，如图 3-260 所示。



图 3-260　新版本中的标签

下面的示例 RFC 包含了一个输入参数“I_Name”和一个表参数“T_Pers”，其中“T_Pers”还包含了其他 4 个字符串类型的参数：ID、NAME、DESC 和 RESERV，如图 3-261 所示。

结构　ZTS_001　激活
短文本　人员结构

Attributes　Components　Entry help/check　货币/数量字段

Predefined Type　1 / 4

Component	Typing Method	Component Type	Data Type	长度	小数位	短文本
ID	1 Types	ID	NUMC	8	0	业务事件的对象标识已提供
NAME	1 Types	NAME	CHAR	35	0	雇员的姓
DESC	1 Types	DESC	CHAR	20	0	Oracle: Time interval when
RESERV	1 Types	ZRESERV	CHAR	200	0	预留字段

图 3-261　RFC 中的参数定义

它的代码逻辑比较简单，作为测试用例，直接返回员工编号“001”、员工姓名“bluse”和员工描述“bluse name”，如图 3-262 所示。

```
FUNCTION ztef_001.
*"----------------------------------------------------------------------
*"*"Local interface:
*"  IMPORTING
*"     VALUE(I_NAME) TYPE  NAME
*"  EXPORTING
*"     VALUE(E_RETURN) TYPE  BAPIRET2
*"  TABLES
*"      T_PERS STRUCTURE  ZTS_001
*"----------------------------------------------------------------------
  CLEAR t_pers[].
  t_pers-id = 001.
  t_pers-name = 'bluse'.
  t_pers-name = 'bluse name'.
  APPEND t_pers.




ENDFUNCTION.
```

图 3-262　RFC 的逻辑处理

1. Web Dynpro 与 RFC

本节主要介绍在 Web Dynpro 中如何使用 RFC 类型的模型。

开发实例：

1）首先创建一个新的 Web Dynpro 工程，并命名为“mydynp03”，如图 3-263 所示。

2）随后创建一个组件，并命名为“Dynp03_01Comp”，在创建向导中同时默认创建视图和窗口，如图 3-264 所示。

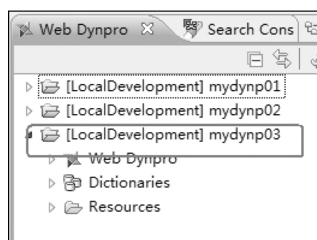


图 3-263　创建新的 Web Dynpro 项目

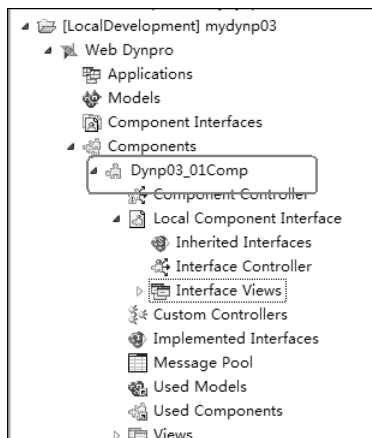


图 3-264　创建新的组件

3）创建组件完毕后，右击“Models”结点，在弹出的快捷菜单中选择创建模型，在随后弹出的对话框中选择第一项“Adaptive RFC 2 Model”，右侧是一些相关的说明，单击“Next”按钮，如图 3-265 所示。

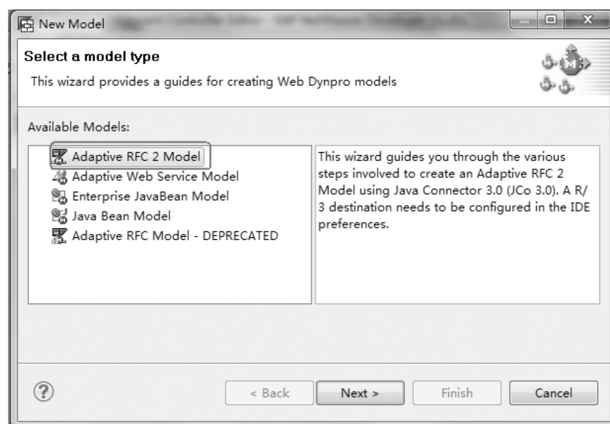


图 3-265　选择导入类型

4）输入导入后的模型名称，以及程序运行时使用的服务端配置的逻辑目的地，如图 3-266 所示。

5）如未配置过导入 RFC 服务器，则单击图 2-266 中的“Configure R/3 Destination”进行服务器参数的配置，以便开发工具连接对应的 ABAP 系统导入 RFC 模型。在弹出的对话

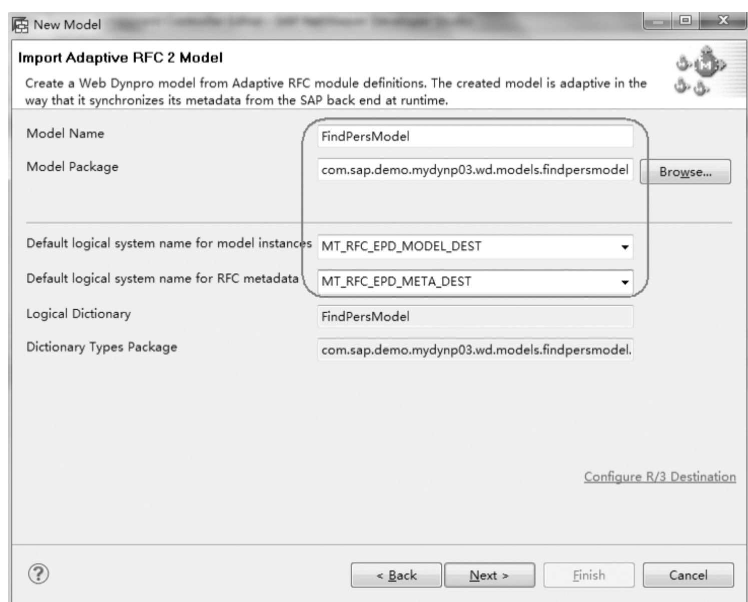


图 3-266　输入模型信息

框中单击“Add”按钮，随后选择“Single Server”（一般需要选择“Load Balancing”负载均衡环境进行配置）并输入服务器的必要信息，如图 3-267 所示。

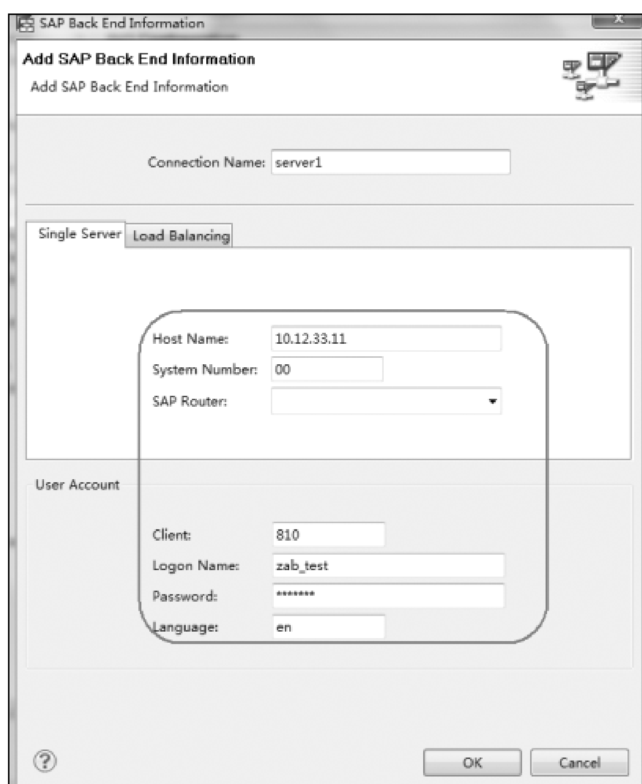


图 3-267　输入服务器连接信息

6）完成服务器配置后，回到导入模型的向导并单击“Next”按钮，如无法连接配置，则系统会弹出相应的提示信息；如出现搜索对话框，则表示连接成功见图 3-268 所示。

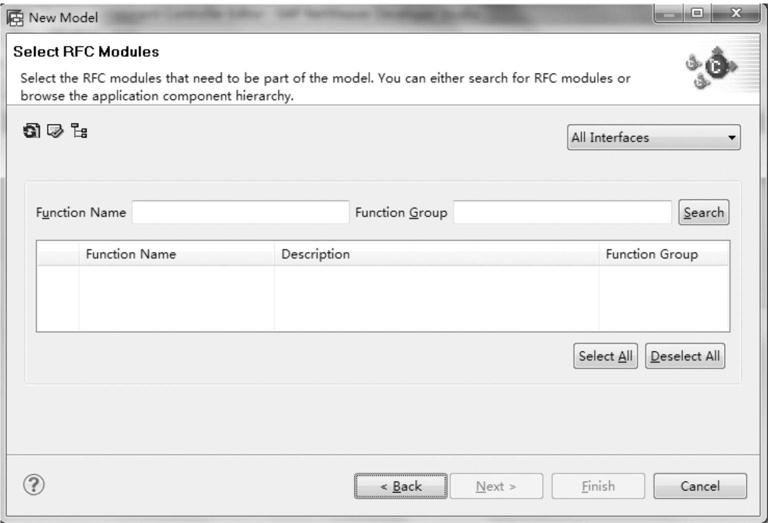


图 3-268　搜索 RFC 对话框

7）输入关键字“ZTE * _001”，单击“Search”按钮，搜索测试使用的 RFC，随后选中需要导入的 RFC 并单击“Next”按钮，如图 3-269 所示。

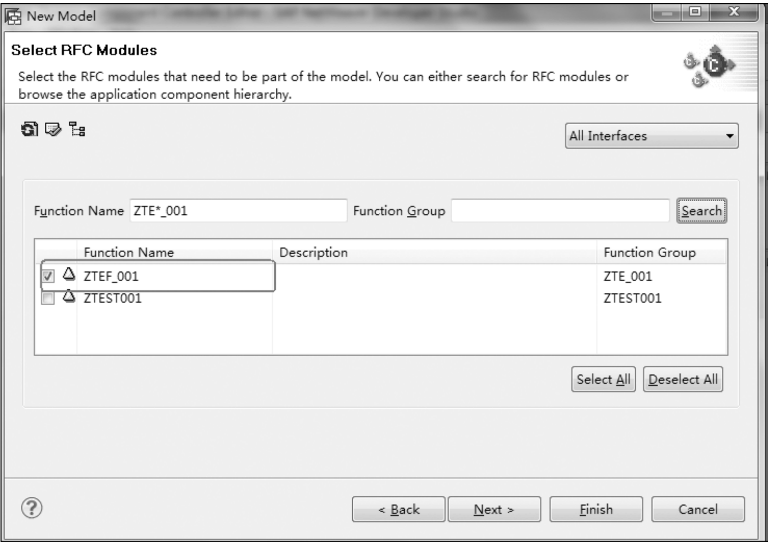


图 3-269　模糊查询对应 RFC

8）在弹出的对话框中会显示导入模型所涉及的表、字段和结构等内容，并生成对应的接口类，单击“Finish”按钮完成模型的导入，如图 3-270 所示。

9）导入完成后就能在“Models”结点下看到导入成功的模型了，如图 3-271 所示。

10）导入完成后，首先需要声明对该模型的使用，进入创建的 Web Dynpro 工程，右击“Used Models”，在弹出的快捷菜单中选择添加之前导入的模型，如图 3-272 所示。

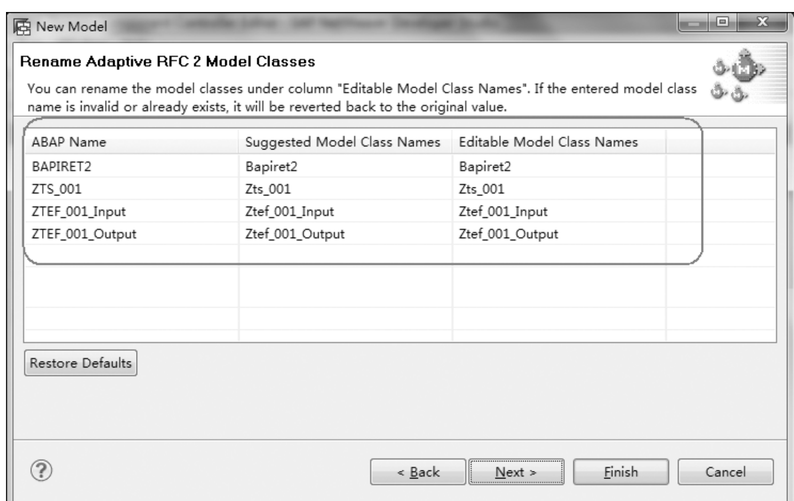


图 3-270　导入内容清单

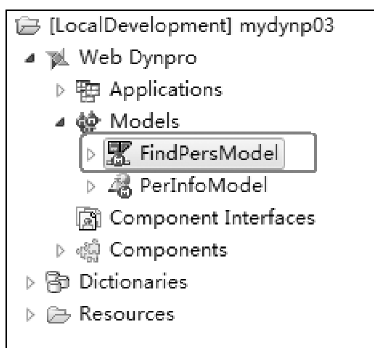


图 3-271　导入后的模型效果

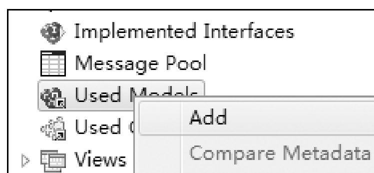


图 3-272　添加使用模型

11）选中“FindPersModel”模型，单击“OK”按钮即完成了添加，如图 3-273 所示。

12）随后右击“Custom Controllers”在弹出的快捷菜单中选择“Apply Template”，自动创建对应的调用方法和上下文，在弹出的对话框中选择“Service Controller”并使用默认名称，单击“Next”按钮，如图 3-274 所示。

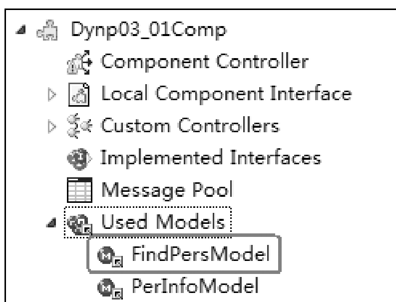


图 3-273　完成模型添加后的效果

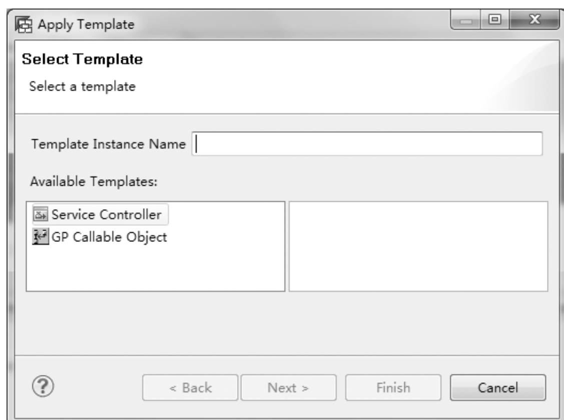


图 3-274　选择模板类型

13）选中要使用的模型，单击“Next”按钮，如图3-275所示。

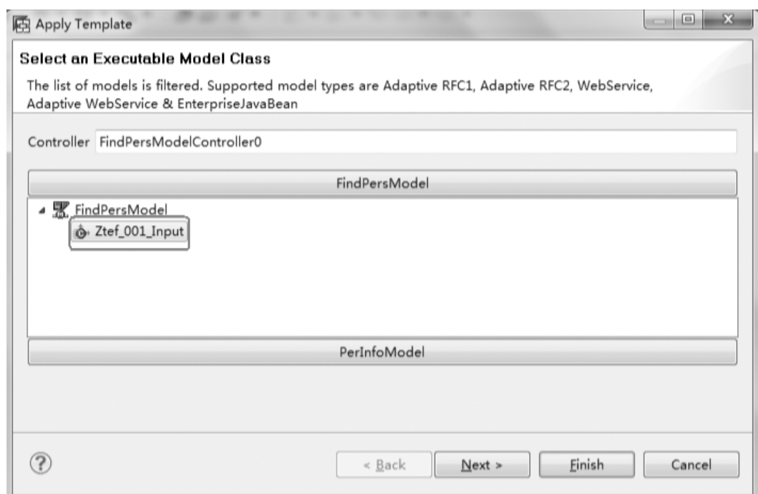


图3-275　选择对应模型

14）选择创建对应的上下文特性并映射的模型的结构和参数，单击“Finish”按钮，如图3-276所示。

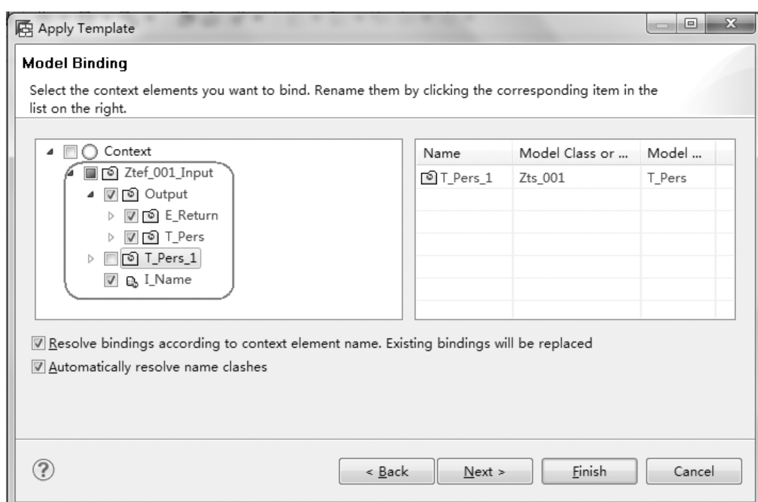


图3-276　确认模型中的参数

15）完成自定义控制器的创建后，保存所有修改即可，如图3-277所示。整个过程从导入模型，声明引用，再到创建自定义控制器，Web Dynpro已经将复杂多样的后台服务统一定制为标有固定输入/输出的模型，且通过自定义控制就可以调用接口中的方法并传递参数，在后续的开发过程中，开发人员就不再需要关心接口服务本身，只专注于自定义控制的方法调用以及上下文映射即可。

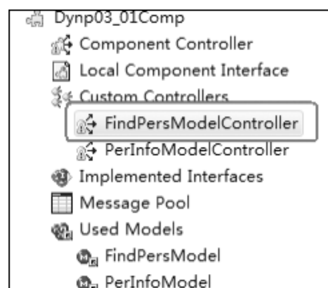


图3-277　创建自定义控制器后的效果

16）同样地，仿照 Web Service 中视图对服务的调用，编写 RFC 的调用过程。首先新建“Dynp03_01Comp_02View”视图，接着在组件控制器中添加上下文的映射，最后在视图的控制器中编写赋值请求数据以及解析响应代码。

17）添加视图到组件控制器，再到自定义控制器的引用关系，如图 3-278 所示。

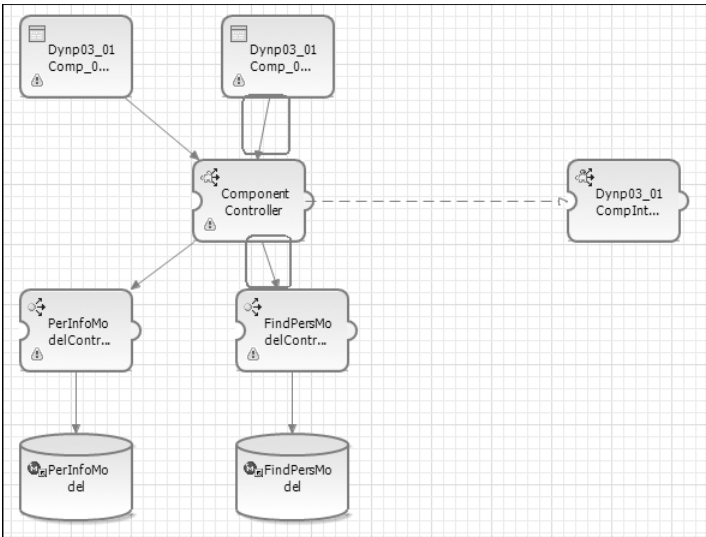


图 3-278　添加引用关系

18）完成从组件控制器到自定义控制器的上下文映射，如图 3-279 所示。

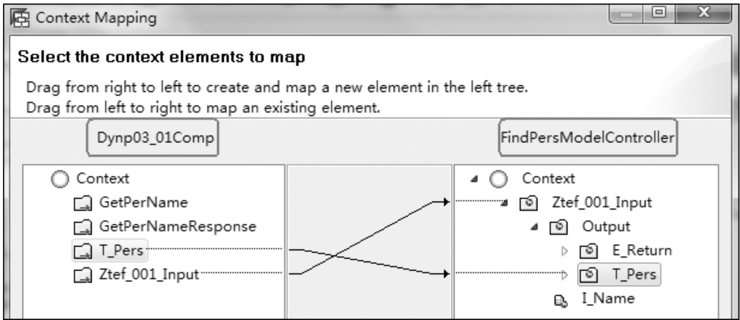


图 3-279　配置上下文映射

19）完成从视图到组件控制器的上下文映射，如图 3-280 所示。

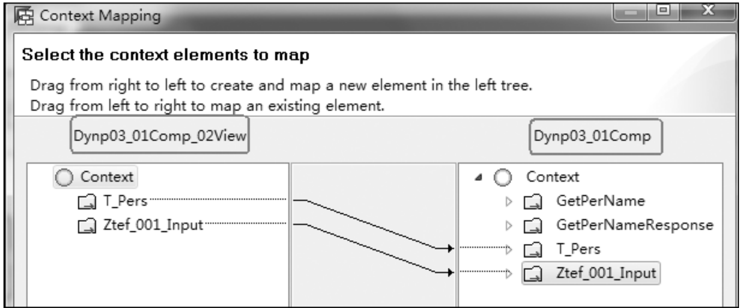


图 3-280　配置上下文映射

20）在组件控制器中添加新的自定义方法“executeFindPer”，如图 3-281 所示。

21）封装方法逻辑：

```
public void executeFindPer( )  {
    //@ @ begin executeFindPer( )
    wdThis. wdGetFindPersModelControllerController( ). executeZtef_001 ( );
    //@ @ end
}
```

22）在视图中添加一个按钮，并创建默认的响应事件，如图 3-282 所示。

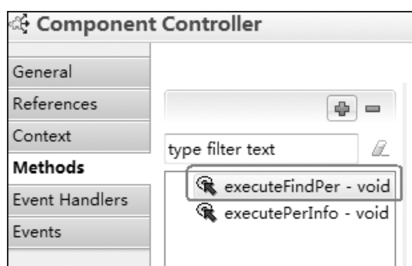


图 3-281　添加自定义方法



图 3-282　添加按钮

23）响应事件的代码逻辑如下：

```
wdContext. nodeZtef_001_Input( ). currentZtef_001_InputElement( ). setI_Name( "王明" );//赋值输
//入参数
wdThis. wdGetDynp03_01CompController( ). executeFindPer( );//执行调用服务方法
```

24）根据映射的上下文返回参数，如图 3-283 所示。

25）使用应用模板方式创建表格元素，如图 3-284 所示。

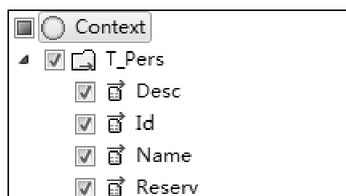


图 3-283　选择上下文

Last name	Event ID	Reserv	Desc
@Name	@Id	@Reserv	@Desc
@Name	@Id	@Reserv	@Desc
@Name	@Id	@Reserv	@Desc

图 3-284　创建表格元素后的效果

26）测试运行。创建 Dynp03_01Comp_02App，编译、部署并运行，运行效果如图 3-285 所示。单击“Button”按钮后就能模拟查询动作，列表显示 RFC 返回结果。



图 3-285　运行效果

27）编辑后台 RFC 函数，增加一条返回记录，如图 3-286 所示。

```
FUNCTION ztef_001.
*"----------------------------------------------------------------------
*"*"Local interface:
*"  IMPORTING
*"     VALUE(I_NAME) TYPE  NAME
*"  EXPORTING
*"     VALUE(E_RETURN) TYPE  BAPIRET2
*"  TABLES
*"      T_PERS STRUCTURE  ZTS_001
*"----------------------------------------------------------------------
  CLEAR t_pers[].
  t_pers-id = 001.
  t_pers-name = 'bluse'.
  t_pers-name = 'bluse name'.
  APPEND t_pers.
  t_pers-id = 002.
  t_pers-name = 'Mike'.
  t_pers-name = 'Mike name'.
  APPEND t_pers.
  t_pers-id = 003.
  t_pers-name = 'Tony'.
  t_pers-name = 'Tony name'.
  APPEND t_pers.


ENDFUNCTION.
```

图 3-286　RFC 代码示例

28）再次单击“Button”按钮，列表更新，如图 3-287 所示。

Button

姓	事件标识	Reserv	Desc
bluse name	00000001		
Mike name	00000002		
Tony name	00000003		

图 3-287　单击“Button”按钮的效果

2. Java 与 RFC

在脱离 Web Dynpro 的情况下，在 Java 代码中可以通过 SAP 提供的 JCO 连接器进行 RFC 的调用。Java 代码调用 RFC 有两种方式，一种是引用第三方 Jar 包，创建 RFC 连接池并维护代码与 RFC 之间连接关系，然而这种方式 RFC 的连接信息和登录信息都未通过服务端配置，维护较为困难；另外一种是引用 SAP 提供的 LIB 类库包，通过 API 调用 RFC 的方式与 Web Dynpro 中自定义控制器的调用的方法相似，并且将程序运行时 RFC 的连接信息通过逻辑目的地配置在服务端维护。前一种方式在网上有很多人分享，本节将对后一种方式的开发过程进行介绍。

更多 SAP JCO 相关的类库和 JAR 包资源，可以通过官方链接进行下载：

http://service. sap. com/connectors

通过一个开发实例了解 Java 调用 RFC 的整个过程：首先创建一个类库工程并包含 SAP JCO 连接器的包文件，随后创建一个 Java 工程并依赖于类库工程，同时在 Java 工程中完成 RFC 的调用逻辑，最后创建一个 Web Dynpro 工程并依赖于 Java 工程，在视图的按钮中直接调用 Java 工程方法即可。

开发实例：

1）首先创建一个测试用的 Web 服务，在 NWDS 中单击右上角的视图切换按钮，切换到 DI 透视图，如图 3-288 所示。

2）右击“MyComponents”自定义的 SC 目录（黄色结点），选择创建一个“Develop-

ment Component”，如图 3-289 所示。



图 3-288　切换到 DI 透视图

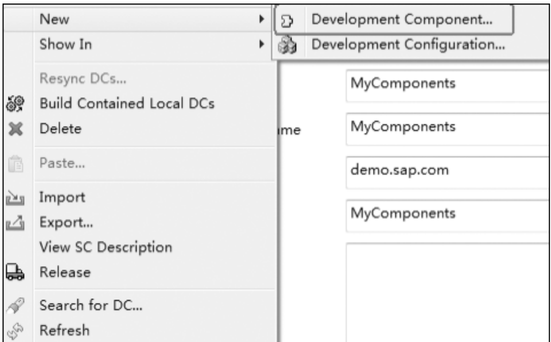


图 3-289　创建开发组件

3）在弹出对话框中选择“External Library”（见图 3-290），单击“Next”按钮。

4）在弹出的对话框中输入工程名称（见图 3-291），单击“Finish”按钮完成类库工程的创建。



图 3-290　选择项目类型

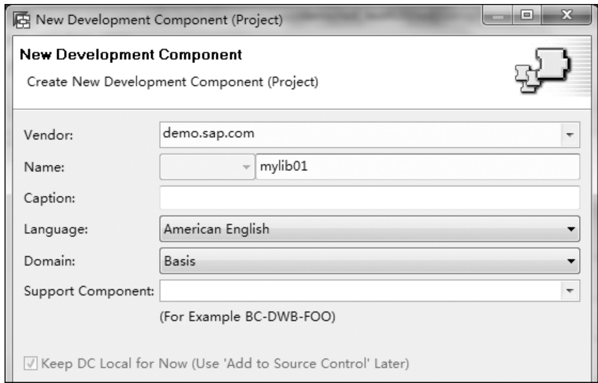


图 3-291　输入工程名称

5）工程创建完成后，将“sapjco3. jar”复制到工程的“libraries”目录中，如图 3-292 所示。

6）继续在 DI 视图中进行公开声明，选中“mylib01”工程，打开右侧的“Public Parts”标签，右击“Manage Entities”，如图 3-293 所示。



图 3-292　添加 JAR 文件

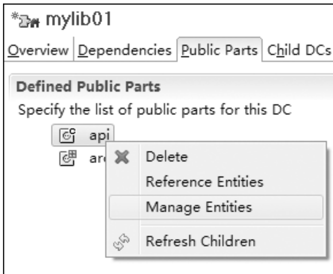


图 3-293　管理实体的菜单选择

7）在随后弹出的窗口中选中“sapjco3. jar”完成添加，在 archives 中也同样添加“sapjco3. jar”的条目（见图 3-294），保存所有修改。

8）接着进行 Java 工程的创建，调用 RFC 测试。再次右击“MyComponents”自定义的 SC 目录，选择“Development Component”，如图 3-295 所示。

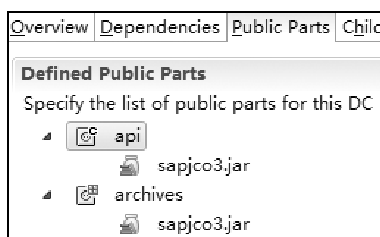


图 3-294　添加后效果

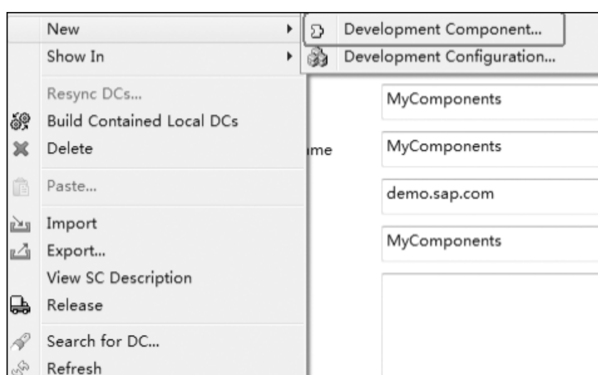


图 3-295　创建开发组件的菜单选择

9）在弹出对话框中选择“Java”类型（见图 3-296），随后单击“Next”按钮。



图 3-296　选择项目类型

10）在弹出的对话框中输入工程名称“myjav03”（见图 3-297），单击“Finish”按钮。

11）继续在 DI 视图中进行类库工程的依赖，选中“myjav03”工程，打开右侧的“Dependencies”标签，单击“Add”按钮，如图 3-298 所示。

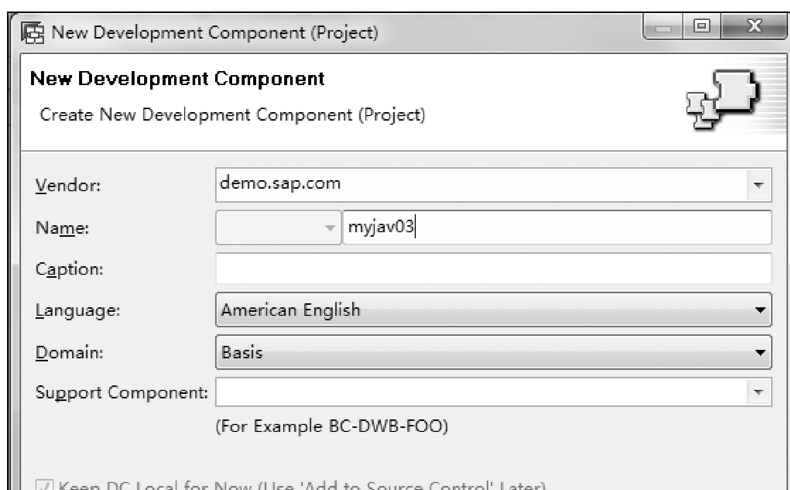


图 3-297　输入工程名称

12）在弹出对话框中找到之前创建的“mylib01”工程并选中，如图 3-299 所示，单击“Finish”按钮。

13）接着进入 Java 工程，创建 RFC 调用类“RFCInvoke”，如图 3-300 所示。

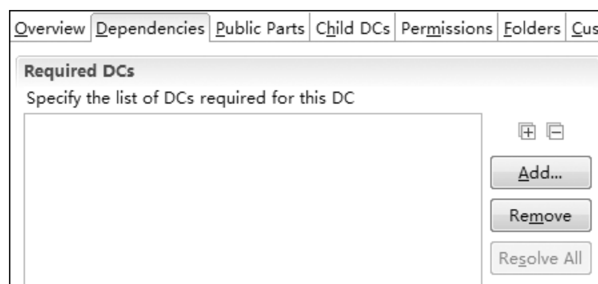


图 3-298　添加项目依赖

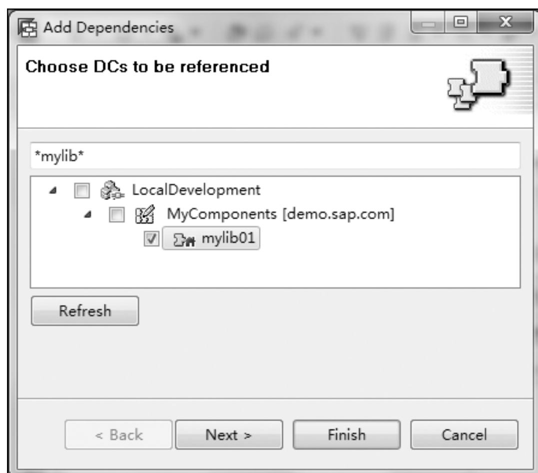


图 3-299　选择依赖工程



图 3-300　创建 Java 类

14）其中调用的参考代码如下：

```
public class RFCInvoke{
    private String DEST = "MT_RFC_EPD_MODEL_DEST";        //Destination Name
    private String RFC_NAME = "ZTEF_001";//RFC Name
      public List<String> callRfc(String i_name)throws Exception{
          JCoDestination destination;
          JCoFunction function;
          // 获取 RFC Destination
          try{
              destination = JCoDestinationManager.getDestination(DEST);
          } catch(JCoException e){
              throw new Exception("DESTINATION ERROR:" + DEST, e);
          }
          // 获取 RFC 函数
          try{
              function = destination.getRepository().getFunction(RFC_NAME);
          } catch(Exception e){
              throw new Exception("RFC NOT FOUND!" + RFC_NAME, e);
```

```
        }
        try{
        // 使用 JcoFunction 对象给服务赋值
            JCoParameterList importParameterList = function
                    . getImportParameterList( ) ;
        // 赋值输入参数
            importParameterList. setValue( "I_NAME" , i_name) ;
        } catch( Exception e) {
            throw new Exception( "" + RFC_NAME + " INPUT PARAMETER ERROR!" , e) ;
        }
        // 执行 RFC
        try{
            function. execute( destination) ;
        } catch( Exception e) {
            throw new Exception( "RFC " + RFC_NAME + "EXECUTE ERROR!" , e) ;
        }
        // 获取返回参数
        try{
            JCoParameterList tableParameterList = function
            . getTableParameterList( ) ;
            JCoTable tPers = tableParameterList. getTable( "T_PERS" ) ;
            if( null != tPers) {
                List list = new ArrayList( ) ;
                for( int i = 0; i < tPers. getNumRows( ) ; i ++ ) {
                    tPers. setRow( i) ;
                    String name = tPers. getString( "NAME" ) ;        // 获取一个参数
                    list. add( name) ;
                }
                return list;
            }
      } catch (Exception e) {
          throw new Exception( RFC_NAME + "" , e) ;
      }
      return null;
   }
}
```

15）进入“Public Parts”，将 Java 工程的类进行公开，如图 3-301 所示。

16）建立 EAR 工程依赖并部署 Java 工程，如图 3-302 所示。

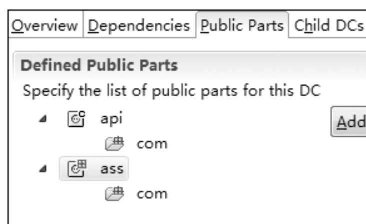


图 3-301　添加公共部分

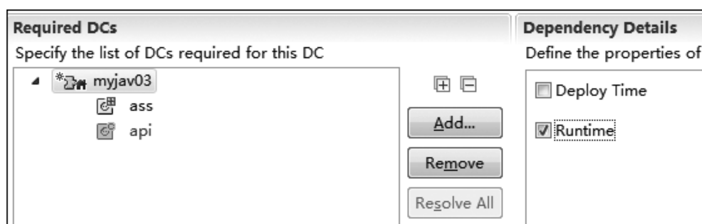


图 3-302　添加工程依赖

17）选择任意一个测试用的 Web Dynpro 工程，在依赖中添加对“myjav03”工程的依赖，如图 3-303 所示。



图 3-303　添加工程依赖

18）保存所有修改并编译后，在视图中创建一个按钮并创建对应动作，如图 3-304 所示。



图 3-304　运行效果

19）在 Java 编辑器中，添加动作的事件处理器代码，创建“myjav03”对象并调用其方法。

```
//@@begin onActionRfc(ServerEvent)
    RFCInvoke invoke = new RFCInvoke();
    try {
        List<String> callRfc = invoke.callRfc("Bluse");
        for (int i = 0; i < callRfc.size(); i++) {
            wdComponentAPI.getMessageManager()
                    .reportSuccess(callRfc.get(i));
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
            wdComponentAPI.getMessageManager()
                .reportException(e.getMessage());
    }
//@@end
```

20）保存所有修改，并将所有工程部署至服务器端后，即可运行，运行效果如图 3-305 所示。

21）单击“RFC”按钮后就能得到 RFC 的返回消息，如图 3-306 所示。



图 3-305　运行效果



图 3-306　单击按钮后的效果

3.7.3　Web 服务模型实例

1. Web 服务的定义与测试

关于 Web 服务（Web Service）的概念这里不再赘述。下面通过 EJB 创建一个简单的 Web 服务。对于 NetWeaver 平台（或者说是 Web Dynpro For Java 对应的 J2EE 应用服务器）而言，所有的应用程序都是通过打包为 EAR 的文件进行部署的。所以，WebService 服务创建包括创建 EJB 工程、创建 EAR 工程、配置 Web 服务等几个步骤。

开发实例：

1）首先创建一个测试用的 Web 服务，在 NWDS 中单击右上角的视图切换按钮，切换到 DI 视图（Development Infratructure perspective）。

2）右击“MyComponents”自定义的 SC 目录（黄色结点），选择创建一个“Development Component”，如图 3-308 所示。

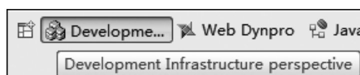


图 3-307

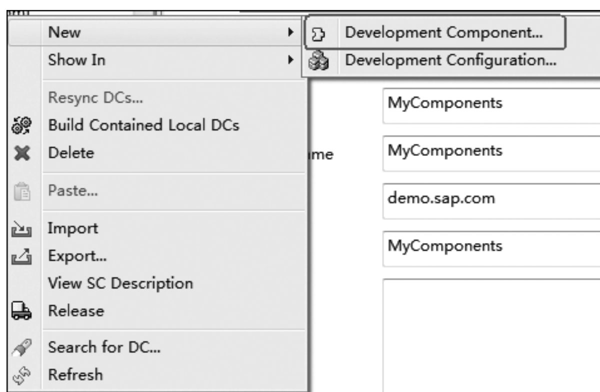


图 3-308　创建开发组件

3）在弹出的窗口中选择对应的 EJB 类型，单击“Next”按钮，如图 3-309 所示。

4）输入自定义 DC 的名称，单击“Next”按钮，如图 3-310 所示。

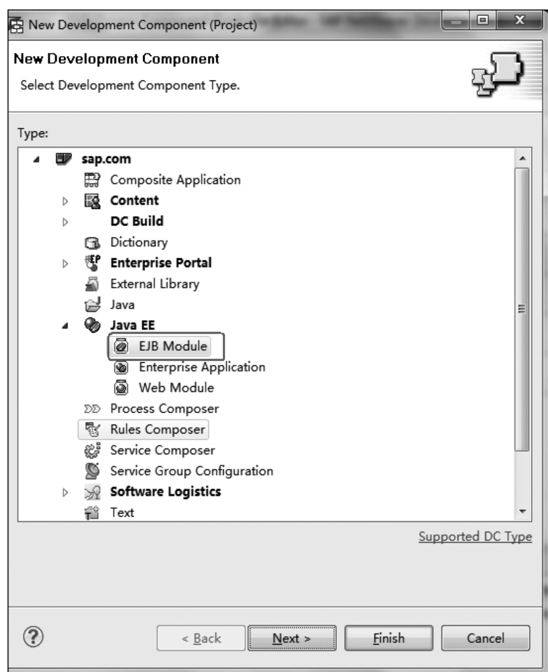


图 3-309　选择项目类型

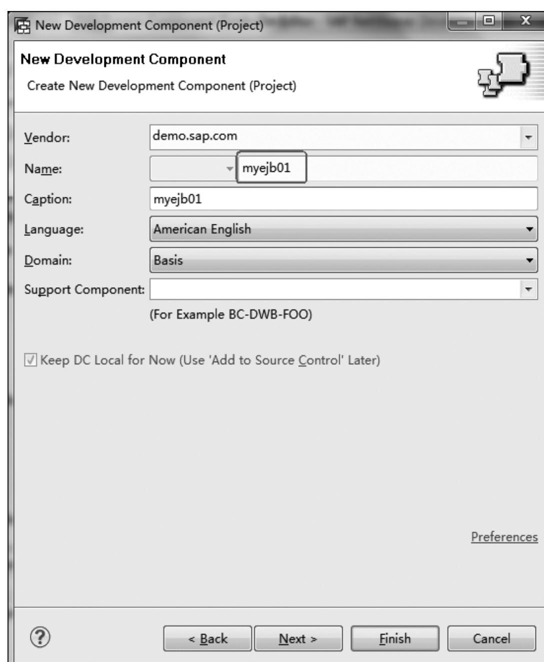


图 3-310　输入项目信息

5）根据实际需求，选择正确的 Java EE 版本（见图 3-311），单击“Finish”按钮。

6）创建完成后，进入 Java EE 视图，右击 EJB 工程，选择创建一个“Session Bean”，如图 3-312 所示。

7）按照命名规范输入自定义的包名和类名（见图 3-313），单击“Finish”按钮，完成 EJB 创建。

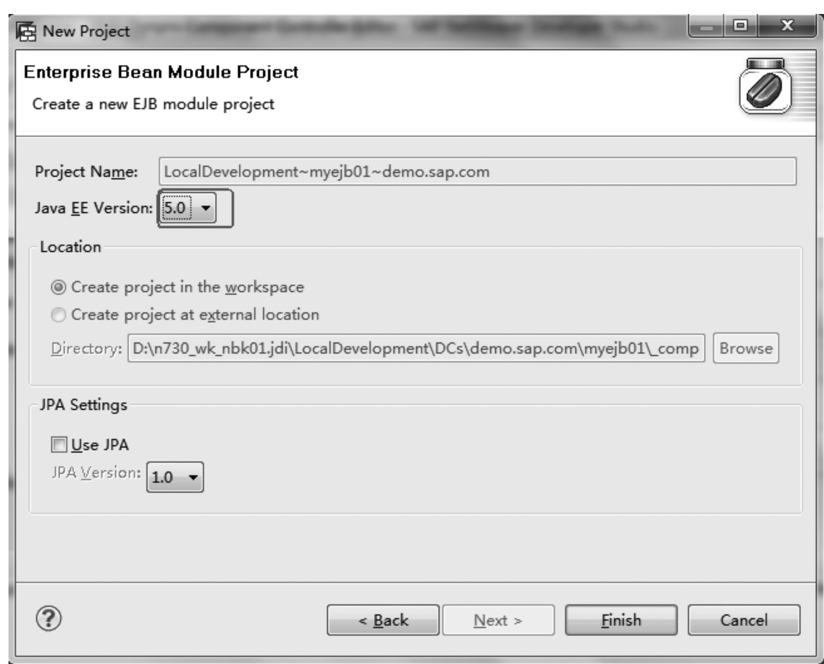


图 3-311　确认编译版本

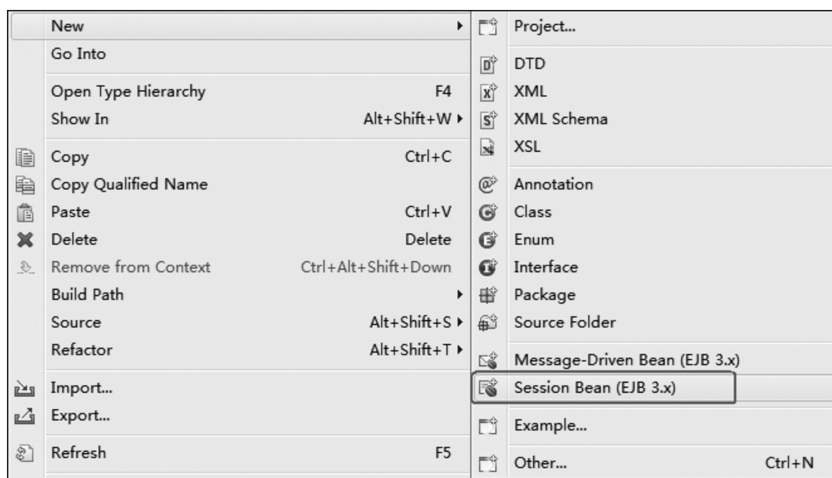


图 3-312　创建 EJB 的菜单选择

8）创建完毕后，可以看到开发工具自动为 EJB 工程生成了一个空的类并实现了一个接口类，如图 3-314 所示。

9）创建一个自定义方法，本例直接返回一个字符串进行测试，加入下列代码，见图 3-315：

```
return "李明";
```

10）打开本地接口类的 Java 编辑器，添加方法的声明，见图 3-316：

```
public String getPerName(String perId);
```



图 3-313　输入项目信息

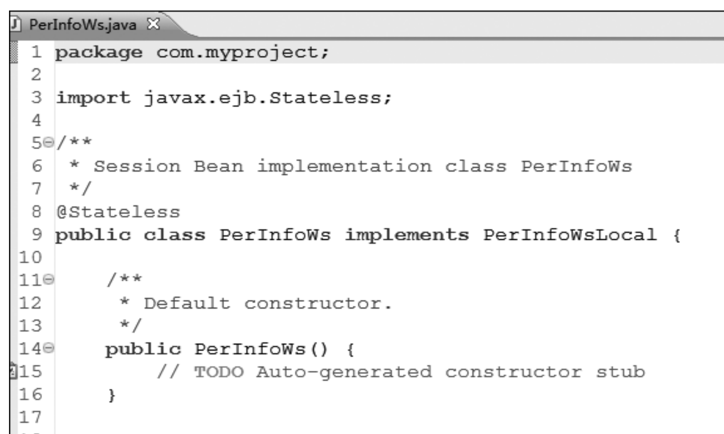


图 3-314　自动生成的类

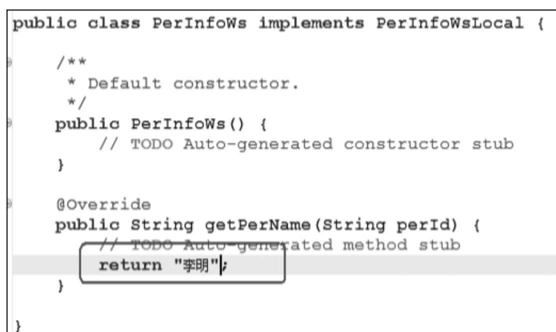


图 3-315　添加代码 1

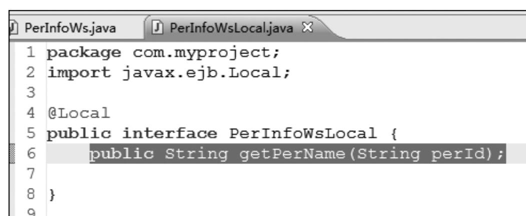


图 3-316　添加代码 2

至此，一个简单的 EJB 创建完毕。创建一个 PerInfoWs 的 EJB，该 EJB 包含对个人信息的操作，同时只添加了一个返回个人姓名的方法。

11）接下来，基于该 EJB 创建对应的 Web Service。再次定位刚刚建立的 Session Bean，右击工程，在弹出的快捷菜单中选择“Web Services”→“Create Web Service”，创建一个 Web Service，如图 3-317 所示。

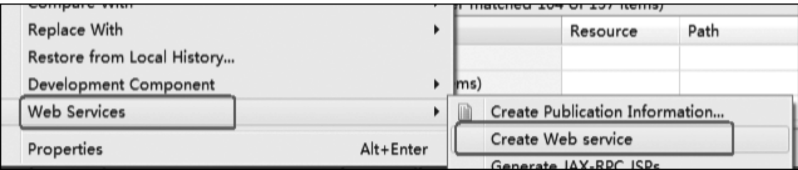


图 3-317　创建 Web 服务的菜单选择

12）在弹出的对话框中，将配置级别的滑块选择“Develop Service”开发级别服务，单击“Next”按钮，如图 3-318 所示。

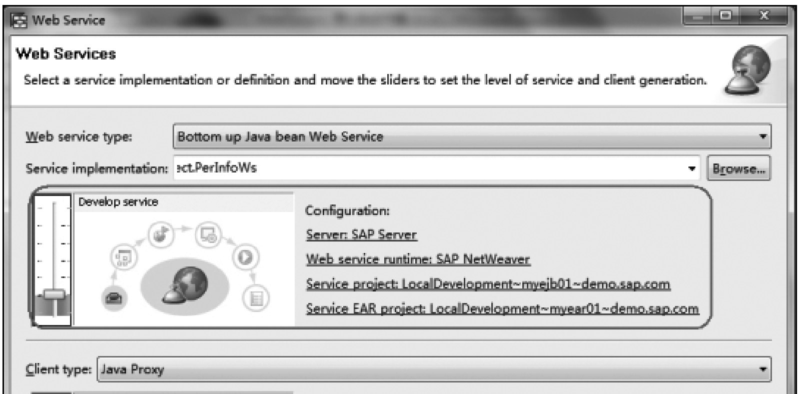


图 3-318　配置 Web 服务属性

13）由于创建 Web 服务需要指定接口，因此单击浏览按钮进行选择，如图 3-319 所示。

14）选择之前在代码中实现并声明的接口方法，单击“OK”按钮，如图 3-320 所示。

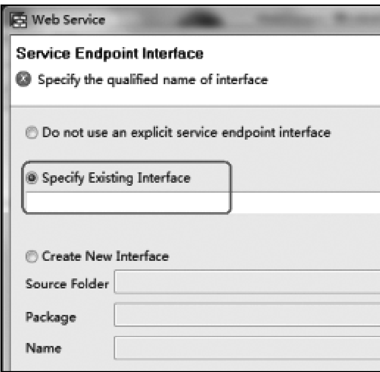


图 3-319　选择接口

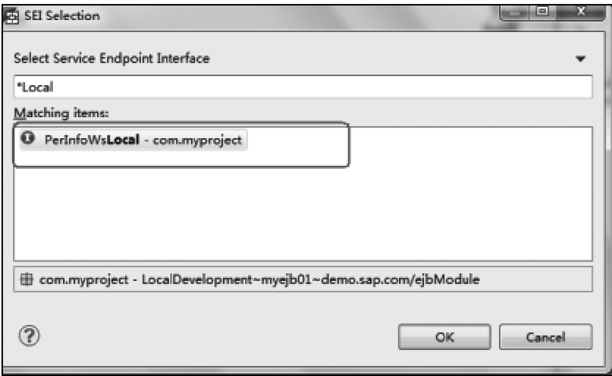


图 3-320　选择接口方法

后面的步骤均使用默认选项直至创建完毕。

15）再次进入 DI 视图，创建一个 EAR 工程，用于依赖 EJB 工程并部署至服务器端。选择新建类型为“Enterprise Application”，如图 3-321 所示，单击“Next”按钮。

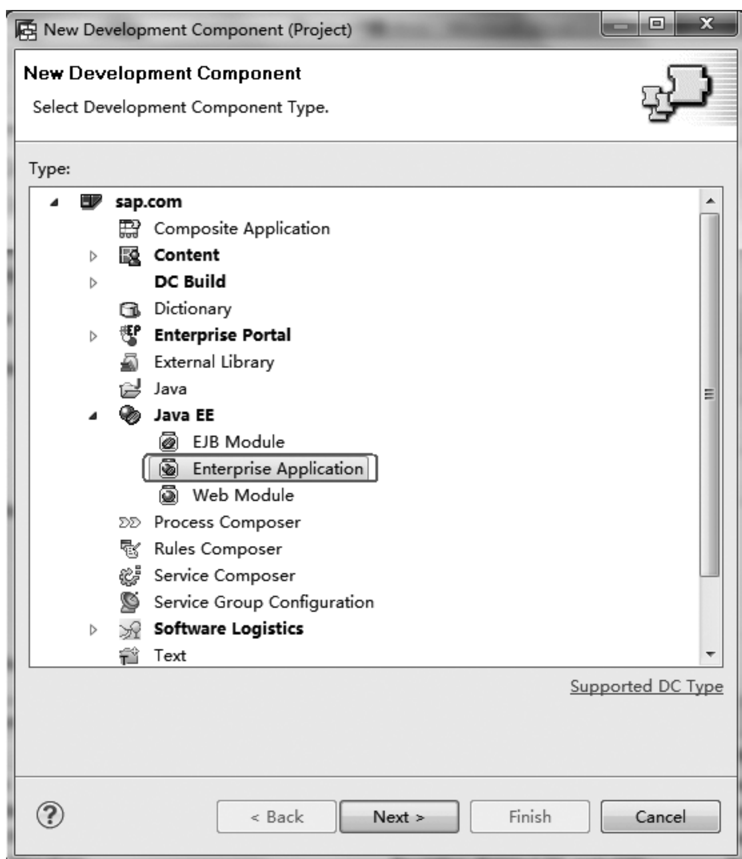


图 3-321　选择项目类型

16）根据实际的命名规范，输入自定义 ear 名称（见图 3-322），单击“Next”按钮。

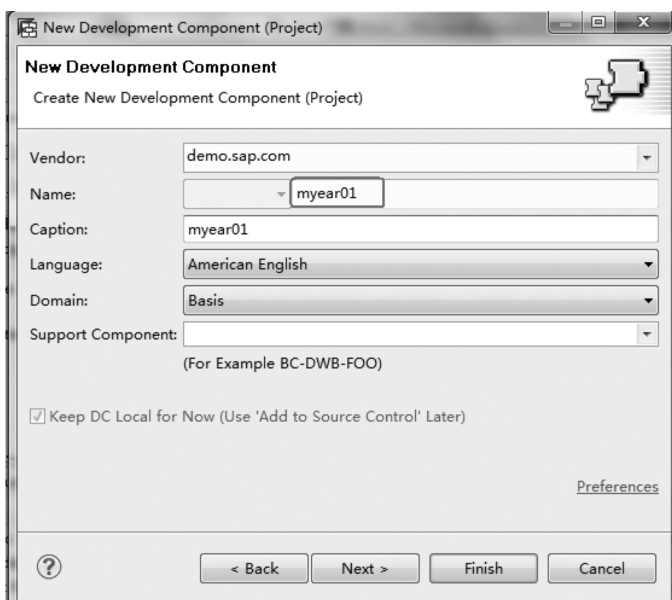


图 3-322　输入项目信息

17）根据实际需求，选择对应的 Java EE 版本，如图 3-323 所示，单击“Next”按钮。

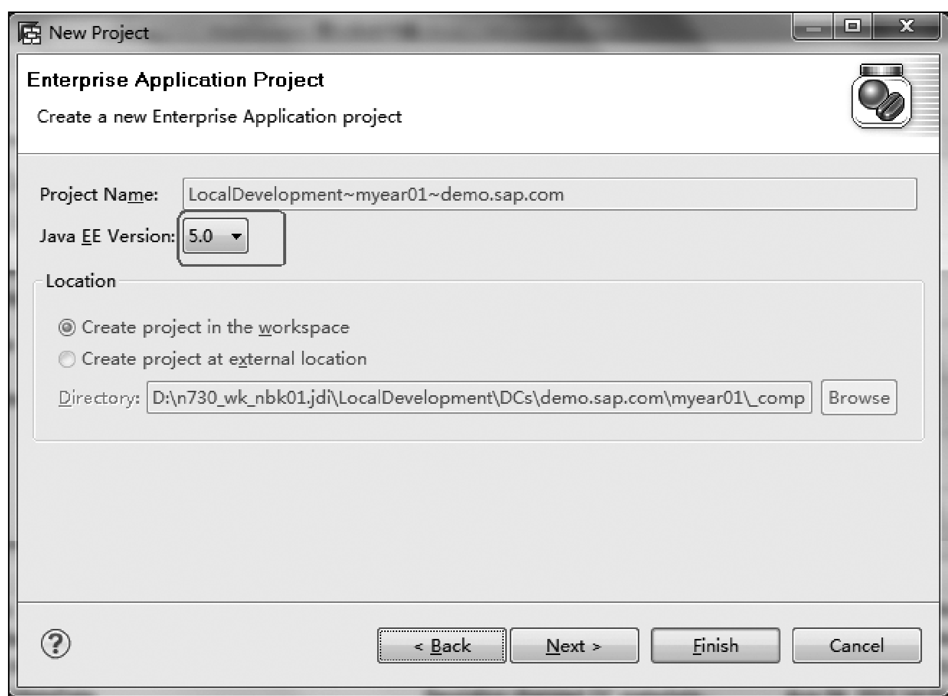


图 3-323　选择对应的编译版本

18）选中对应引用的 EJB 工程（见图 3-324），单击“Finish”按钮。

19）保存所有修改，完成两个工程的创建，如图 3-325 所示。

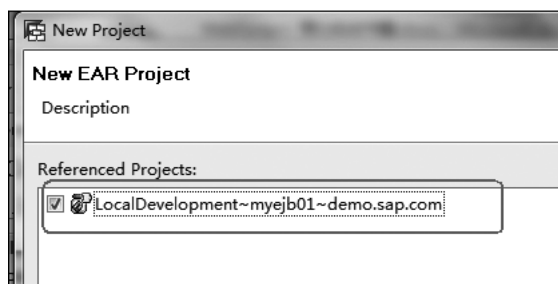


图 3-324　选择引用的 EJB 项目

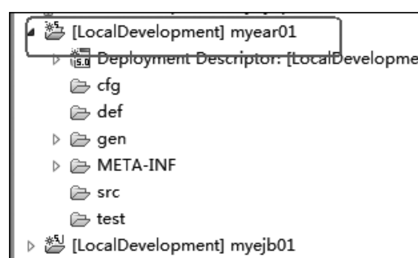


图 3-325　创建后的效果

20）通过开发组件的菜单进行编译并部署 EAR 工程，将创建的 Web Service 发布到服务器。右击 EAR 工程名，选择“Development Component”→“Build”进行编译，编译完成后在同一菜单中选择“Deploy”进行部署，如图 3-326 所示。

21）下面验证一下 Web 服务是否部署成功。用管理员账号登录服务器 NWA 管理页面，并进入“NWA”→“Configuration”→“Connectivity”→“Single Service Administration”，如图 3-327 所示。

22）在搜索条件中输入“*PerInfo*”，单击“Go”按钮，如图 3-328 所示，搜索创建的 Web 服务。

23）已成功找到对应的 Web 服务，证明部署成功，如图 3-329 所示。

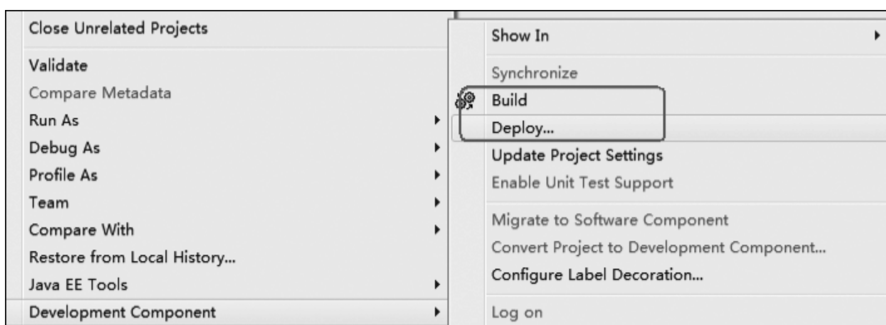


图 3-326　部署应用的菜单选择

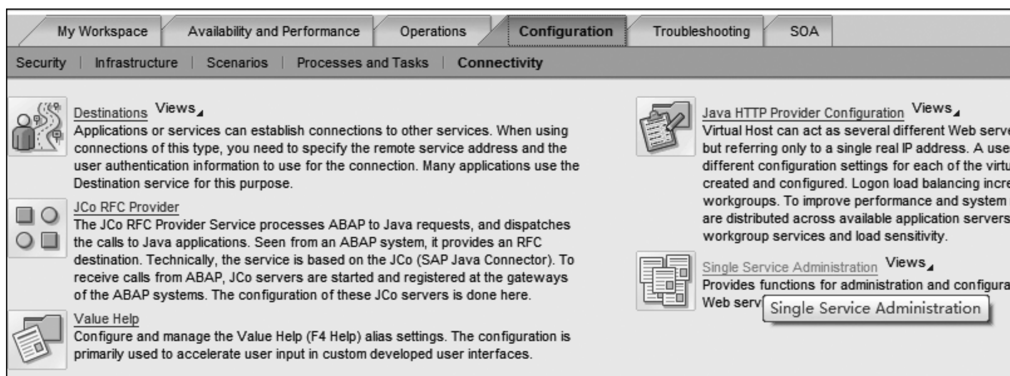


图 3-327　NWA 平台中的独立服务管理

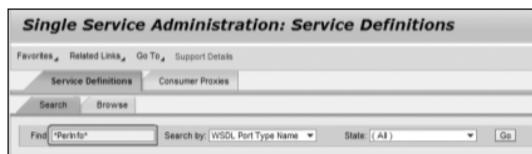


图 3-328　搜索对应服务

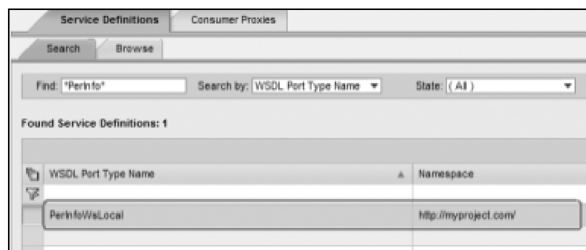


图 3-329　搜索服务结果

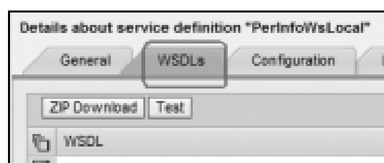


图 3-330　获取 WSDL 信息

24）在下面的 WSDLs 中，可以看到 Web 服务对应的 WSDL 地址，单击“Test”按钮对功能进行测试，如图 3-330 所示。

25）在测试向导中选择需要测试的方法，选择自定义创建的“getPerName”方法（见图 3-331），单击

“下一步”按钮。

26）由于方法不需要输入参数（见图 3-332），因此直接单击“Next”按钮即可。



图 3-331　测试对应服务



图 3-332　输入参数值

27）运行结果右侧显示了服务正确返回的执行结果，功能测试完毕，如图 3-333 所示。

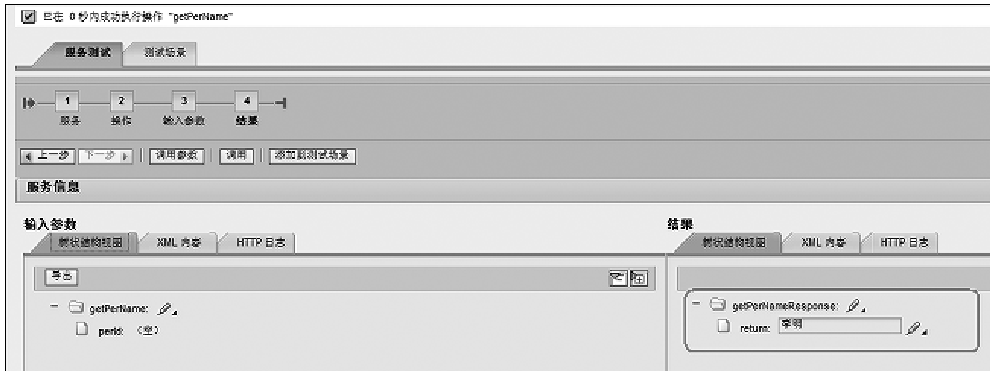


图 3-333　运行效果

通过上面的验证，说明 Web 服务已经正确创建并发布至服务器。

其实创建 Web 服务的方式有很多，对 Java 来讲就有很多，更何况延伸至不同的平台都有对应的发布 Web 服务的方式。这里介绍的是通过 NWDS 提供的 Web 服务的创建向导快速创建 Web 服务的方法，并且服务端也有针对发布、测试、配置的一整套环境，使得 Web 服务的使用更加地灵活和方便。

2. Web Dynpro 与 Web 服务

上一节通过 NWDS 创建了一个简单的 Web 服务，下面通过一个开发实例重点介绍如何在 Web Dynpro 中通过模型的创建和配置进行 Web 服务的调用。

开发实例：

1）使用现有的 Web Dynpro 工程，或重新创建一个简单工程，如图 3-334 所示。本例以之前创建的 mydynp03 为例。

2）右击“Models”，在弹出的快捷菜单中选择“Create Model”，如图 3-335 所示。

3）在弹出的窗口中选择对应的模型类型，这里选择第二项“Adaptive Web Service Model”，单击“Next”按钮，如图 3-336 所示。

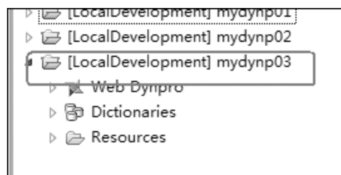


图 3-334　原 Web Dynpro 项目目录

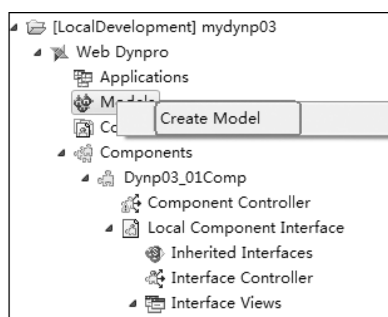


图 3-335　创建模型的菜单选择

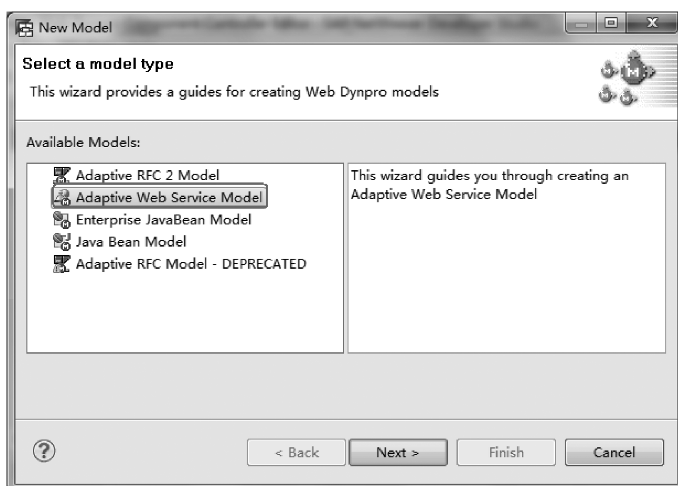


图 3-336　选择模型类型

4）输入自定义模型名称，选中“Remote Location/File System”单选按钮作为 WSDL 描述文件的源文件，单击“Next”按钮，如图 3-337 所示。

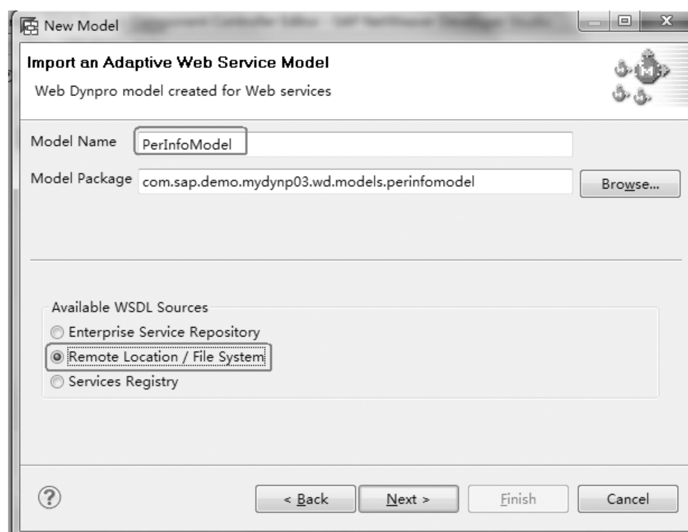


图 3-337　输入模型信息

5）输入创建的 Web 服务对应的 WSDL 地址（在 Service Definitions 中下方的“WSDLS”页签中可以看到），粘贴的同时开发工具会立即对服务做连通性测试。测试成功后，单击“Next”按钮，如图 3-338 所示。

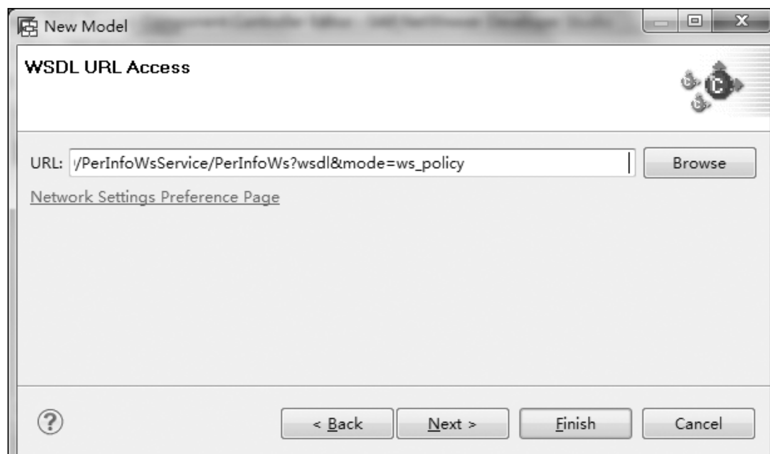


图 3-338　输入服务地址

6）接着进行运行时间相关配置，选中“No service group configuration”单选按钮，不需要配置服务组，单击“Next”按钮，如图 3-339 所示。

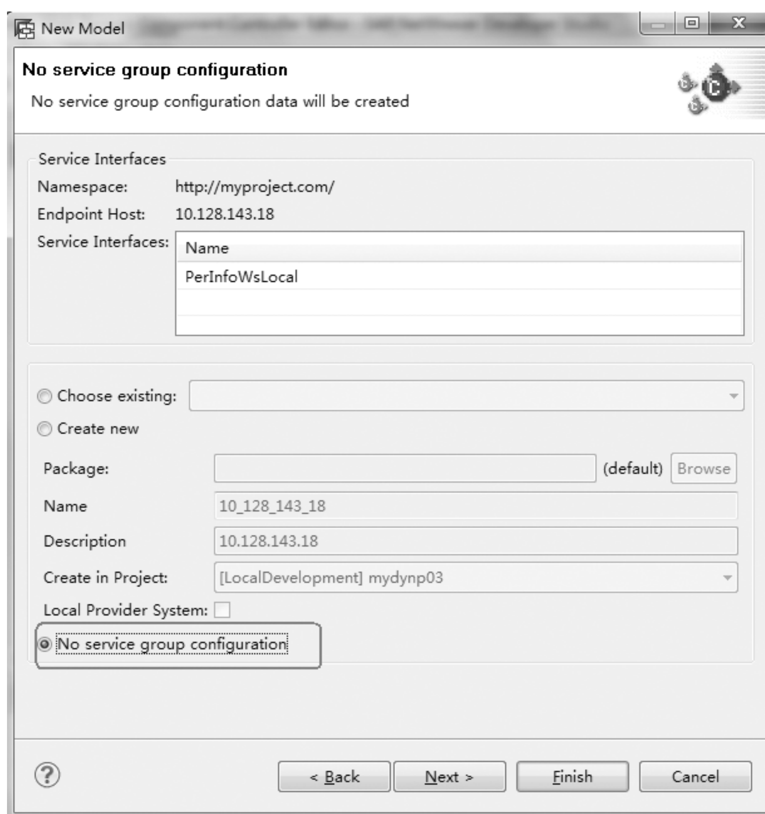


图 3-339　确认服务配置

7）输入逻辑目的地名称，输入“MT_WS_PERINFO_DEST”（常用的命名规范为系统名+服务类型标识+服务标识+目的地，根据实际系统配置进行填写），单击“Next”按钮。此处 MT 默认为系统名，服务类型为 Web Service，服务标识或描述为 PERINFO，并且 Metadata 与 Execution 采用同一个目的，如图 3-340 所示。

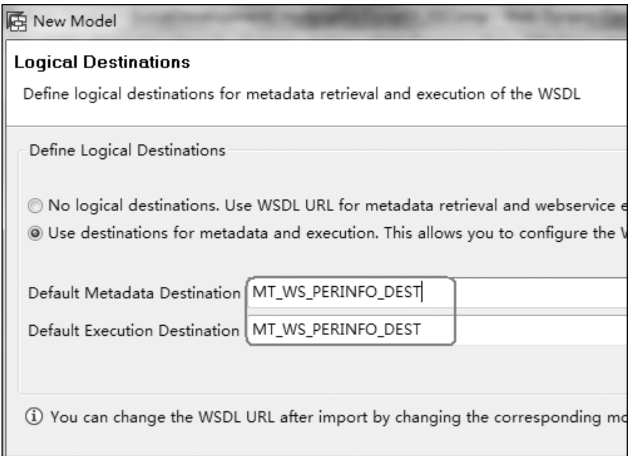


图 3-340　配置目的

如果仅采用 URL 做为源数据的目的地去执行，当服务的地址或端口变化后，则需要重新导入模型来更新实际开发时的 URL 远程访问地址。然而实际上对于正在运行的系统来讲，通过修改程序的方式去应对服务端接口的变化显然是不合理的，一方面是带来了较大的维护开发量，另外一方面如果在用户使用中去重新布署程序也会带来系统运行或数据丢失的风险。SAP 使用服务器端配置的目的地就能够很好地解决这个问题，当运行时地址发生变化时，直接修改配置内容的 WSDL 地址即可。

8）根据需要对导入的模型类进行重命名，跳过则直接单击“Next”按钮，如图 3-341 所示。

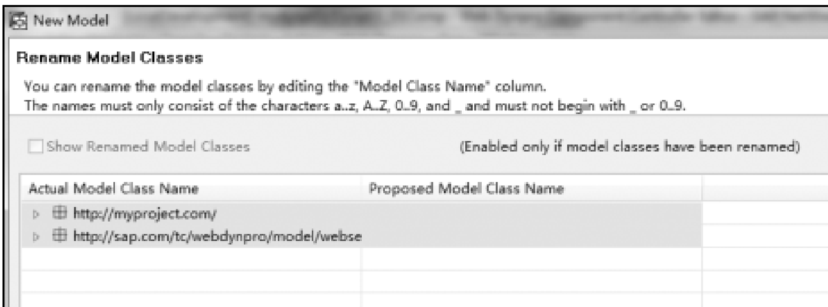


图 3-341　确认导入内容

9）在弹出的对话框中，可以看到导入日志，也就是看到对应 Web 服务接口在 Web Dynpro 模型中生成的接口结构。根据接口方法的输入/输出，同时也会生成对应的请求与响应结构，请求结构中包含一个参数为 PerId（方法的输入参数名），响应的结构中包含一个参数为 Return（方法返回 String 类型的数据，系统默认命名为 Return）。单击“Finish”按钮完成模型的导入配置，如图 3-342 所示。

10）完成配置后，就可以在工程的目录中看到对应导入的模型以及模型类结构，如图 3-343 所示。

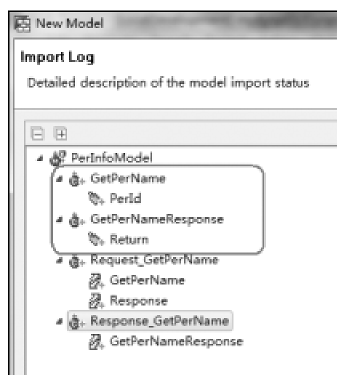


图 3-342　导入日志查看

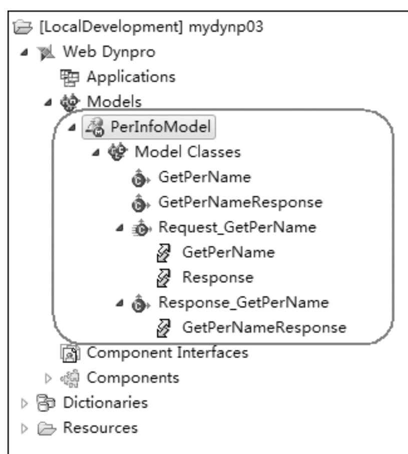


图 3-343　导入后的效果

11）在导入模型完成之后，需要创建对模型接口的调用，即创建自定义控制器。

首先需要添加对导入模型的引用：右击组件中的“Used Models”，在弹出的快捷菜单中选择“Add”，如图 3-344 所示。

由于之前的步骤中导入的模型是整个 Web Dynpro 工程可以使用的，只有在需要使用的组件中添加特定模型的引用，才能在组件中进行模型接口的调用。

12）在弹出的对话框中选择之前导入的模型，单击“OK”按钮，如图 3-345 所示。

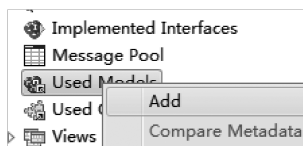


图 3-344　添加使用模型

13）添加引用之后，需要创建一个自定义控制器。在组件中右击“Custom Controller”，在弹出的快捷菜单中选择“Apply Template”，如图 3-346 所示。

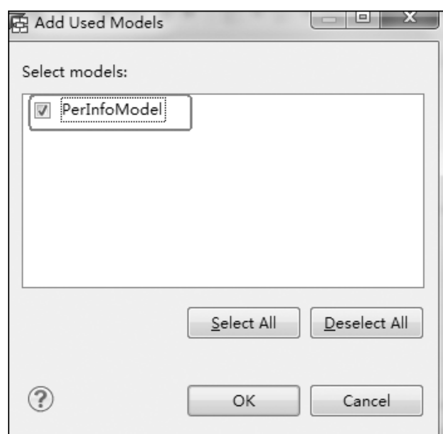


图 3-345　选择模型

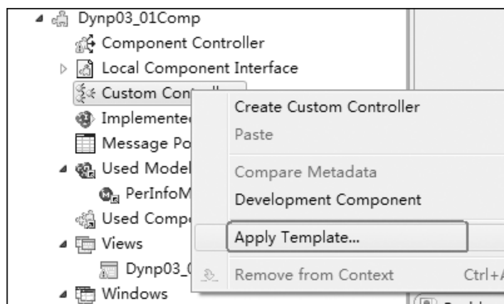


图 3-346　应用模板的菜单选择

14）选择“Service Controller”服务控制器，单击“Next”按钮，如图 3-347 所示。

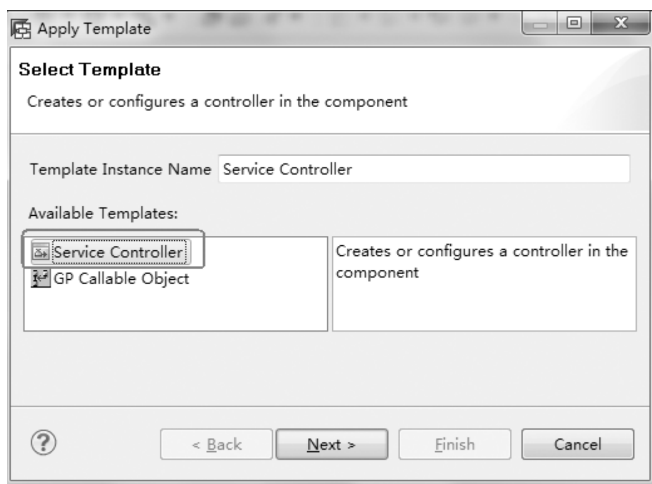


图 3-347　选择模板类型

15）选择之前导入的“PerInfoModel”模型的请求方法，单击“Next”按钮，如图 3-348所示。

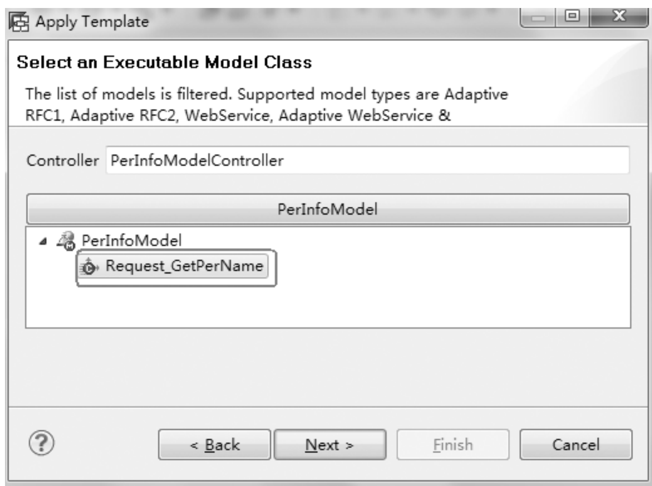


图 3-348　选择对应的模型

16）对模型字段自动生成的自定义控制器中的上下文进行具体配置，可以选择所有字段也可以选择部分字段进行创建。仔细分析可以看到，“Request_＊”作为根结点，该结点下面的“Response”为方法返回生成的响应数据结构，与“Response”同级的为方法输入生成数据结构，因此不管是赋值还是解析，都是对“Request_＊”下面数据节点的操作。在本例中，选中所有的结点作为自定义控制器中的上下文，并同时完成映射配置，随后单击“Next”按钮，如图 3-349 所示。

其实在这一步配置中，与之前视图和控制器的上下文映射非常相似，在这里完成的则是自定义控制器的上下文与导入模型的数据结构进行映射。

17）默认可以同时创建模型的执行方法，即触发接口调用，输入方法名称，随后单击“Finish”按钮，如图 3-350 所示。

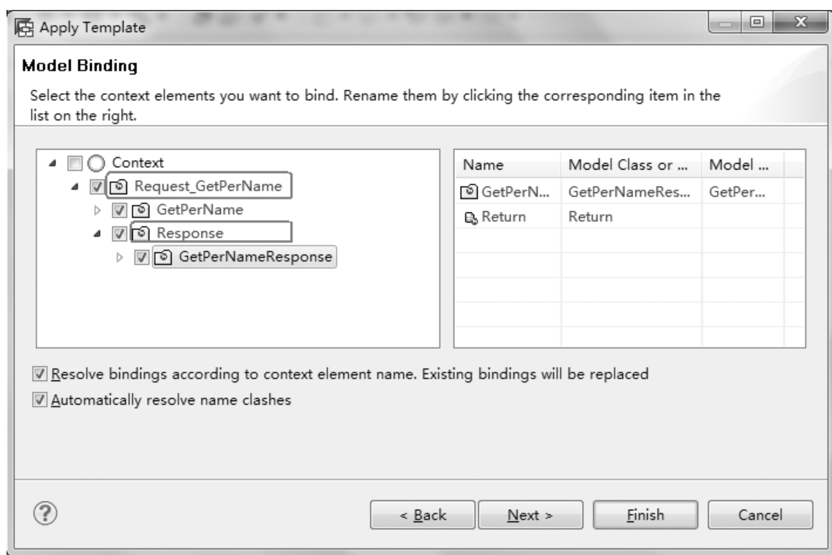


图 3-349　确认应用结点

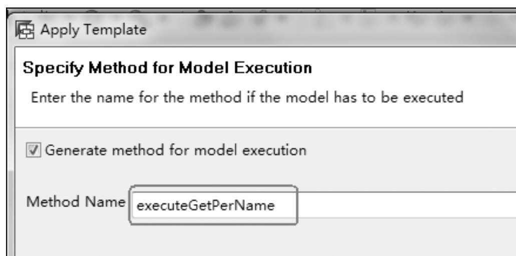


图 3-350　输入方法名称

18）完成配置后，记得保存所有修改，检查编译错误，如图 3-351 所示。完成了自定义控制器的创建，在组件的模型编辑器中看到，增加了自定义控制器和导入模型两个对象，同时它们之间也已经完成了映射配置。

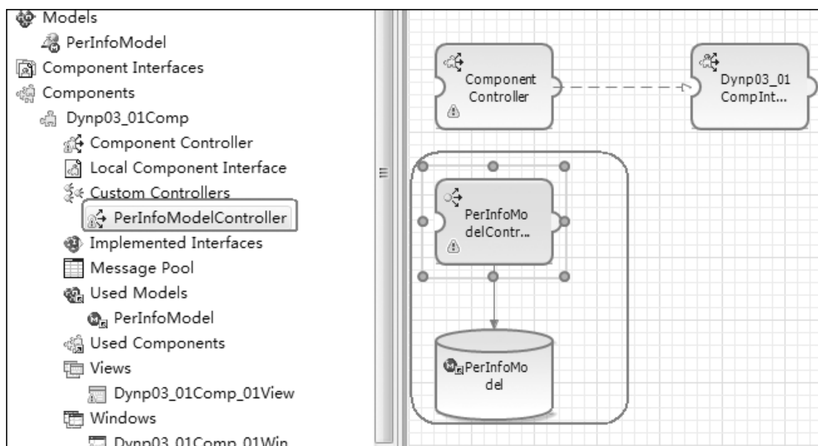


图 3-351　完成配置后的效果

19）接着需要从视图中对接口进行调用。首先将自定义控制器的方法封装至组件控制器中，通过数据链接添加组件控制器对自定义控制器的引用，如图 3-352 所示。

20）将两个控制器的上下文进行映射，如图 3-353 所示。

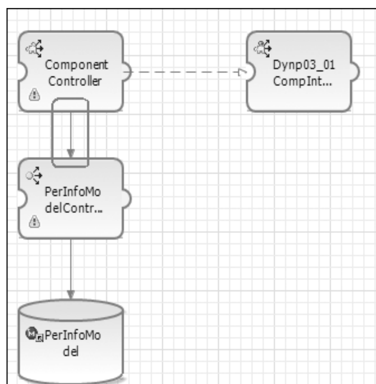


图 3-352　引用关系

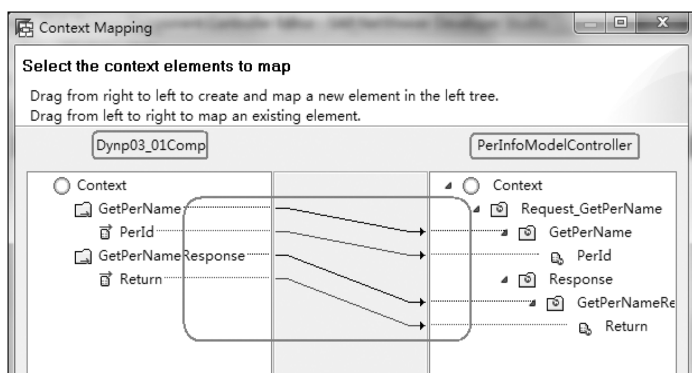


图 3-353　上下文映射关系

21）在组件控制器新增自定义方法“executePerInfo”，添加后保存修改，如图 3-354 所示。

22）保存之后，双击方法进入代码编辑器，添加以下代码：

```
//获取自定义控制器的实例,并执行自定义控制器中的“execute”方法
wdThis.wdGetPerInfoModelControllerController().executeGetPerName();
```

23）将视图的上下文与组件控制器上下文进行映射，如图 3-355 所示。

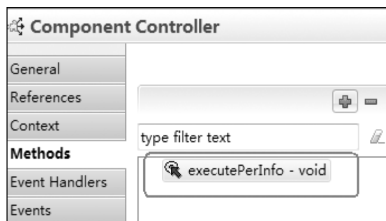


图 3-354　添加自定义方法

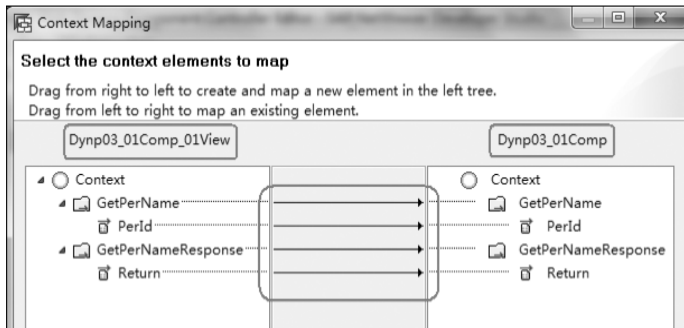


图 3-355　添加上下文映射

24）打开“Dynp03_01Comp_01View”视图，在“Outline”元素清单中右击根结点，选择“Apply Template”，如图 3-356 所示，选择按钮类型并单击“Next”按钮至结束。

25）完成按钮创建后，保存所做的修改。随后双击按钮进入代码编辑器，如图 3-357 所示。

在按钮的事件处理器方法中添加赋值与打印代码：

```
wdContext.nodeGetPerName().currentGetPerNameElement().setPerId("0010000");//赋值请求参数
wdThis.wdGetDynp03_01CompController().executePerInfo();//执行调用 Web Service 接口的方法
String return1 = wdContext.nodeGetPerNameResponse().
currentGetPerNameResponseElement().getReturn();//获取返回参数
wdComponentAPI.getMessageManager().reportSuccess(return1);//打印返回值
```

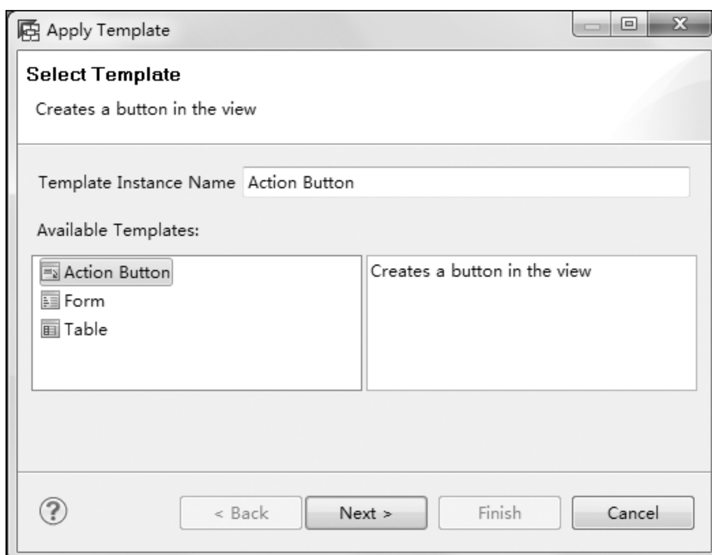


图 3-356　选择模板类型

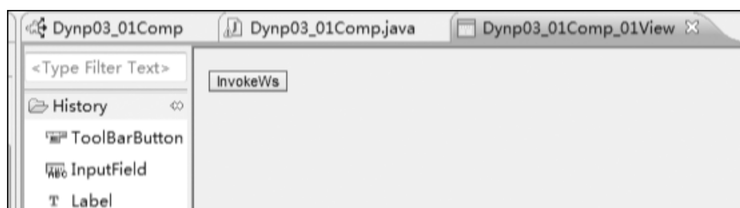


图 3-357　创建按钮后的效果

26）至此，所有配置和代码工作都已完成，保存所有修改并编译部署至服务器，如图 3-358 所示。

27）单击“InvokeWs”按钮后，就能在页面下方看到 Web 服务返回的姓名信息，如图 3-359 所示。



图 3-358　运行效果

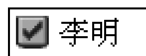


图 3-359　单击按钮效果

关于运行时的目标模板配置说明如下。

上述创建模型时，在配置本地目标的步骤输入了“MT_ WS_ PERINFO_ DEST”，这个目标是需要在服务器端提前进行配置的，程序运行时模型的元数据运行程序也会通过配置的目标查找对应的接口、传递参数以及获取返回信息。

1）使用管理员账户登录到服务器 NWA 平台，单击“SOA”→“Technical Configuration”命令，随后单击“Destination Template Management”目标模板管理链接。

2）进入页面后，单击“New”按钮，新建一个目标输入的信息，如图 3-360 所示。

3）单击“Next”按钮，在下一步中输入服务验证信息。由于之前在创建 Web Service 时并未添加验证，因此此处选择“None”，如图 3-361 所示。

图 3-360　输入目标信息

图 3-361　选择验证方式

4）完成所有信息填写后，单击“Finish”按钮即可。

3. Java 与 Web 服务

Java 访问 Web 服务的方式有很多，目前比较成熟的框架包括有 XFire、AJAX 等。传统 Java 工程通过引入这些框架对应的第三方 Jar 包以及它们提供的 API 便可以完成对 Web 服务的调用，但前提是需要自己编写调用逻辑。在实际的项目开发中，Web 服务的接口往往是比较复杂的，尤其对于嵌套的集合结构类型，客户端在不管构造赋值请求数据还是解析响应都是一个非常复杂而量大的过程，这样无疑降低了开发者的开发效率，同时也增加了接口双方调试的难度。而 Web Dynpro 在调用 Web 服务时自动生成本地客户端代理，并且提供的接口报文日志方便在接口调试时做数据跟踪，以及运行时的配置来应对服务端地址或端口变化的情况。

开发实例：

1）切换到 DI 视图，右击自定义的 SC 目录，选择“Development Component”，如图 3-362 所示。

2）在弹出的窗口中选择对应的 EJB 类型，单击“Next”按钮，如图 3-363 所示。

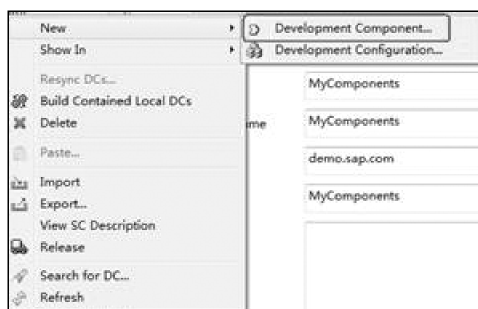


图 3-362　创建开发组件的菜单选择

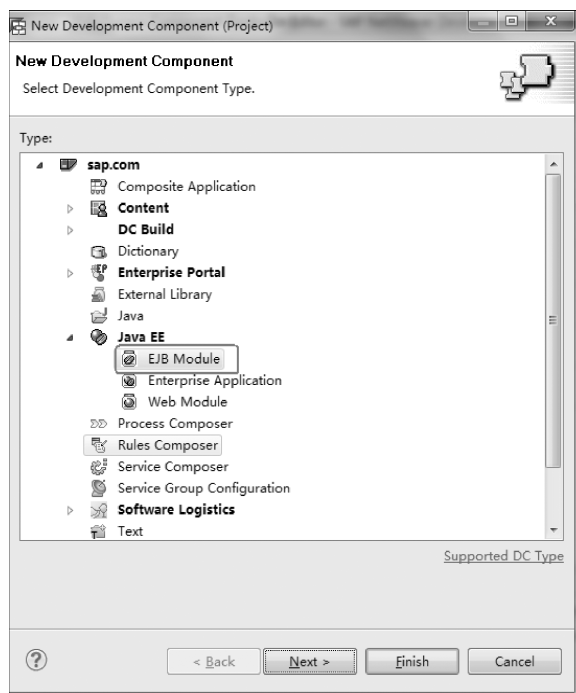


图 3-363　选择项目类型

3）输入自定义 DC 的名称，单击“Finish”按钮，如图 3-364 所示。

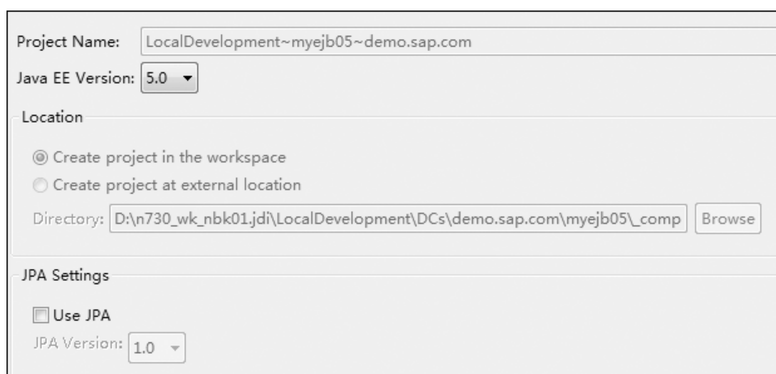


图 3-364　选择编译版本

4）创建工程完毕后，展开工程，右击“ejbModule”中的“META - INF”，并选择“Import”导入，导入类型选择“Web services”中的“WSDL”，单击“Next”按钮，如图 3-365 所示。

5）描述文件源选择“Remote Location/File System”单选按钮，如图 3-366 所示，单击“Next”按钮。



图 3-365 选择服务类型

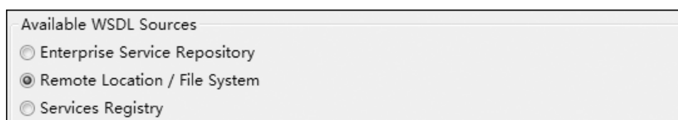


图 3-366 选择源文件类型

6）输入创建的 Web Service WSDL 地址，该地址通过服务器中 NWA 管理员平台的“Service Definitions”服务目的地进行查看，单击“Finish”按钮，如图 3-367 所示。

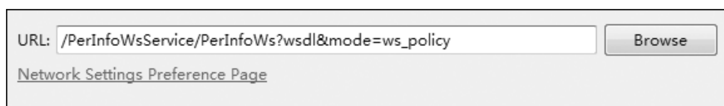


图 3-367 输入服务地址

7）完成 WSDL 文件的导入之后，可以看到在文件夹中已导入了对应的文件，如图 3-368 所示。

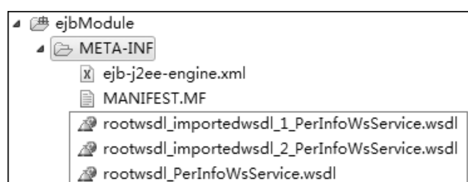


图 3-368 导入后的效果

8）右击导入的 rootwsdl_PerInfoWsService. wsdl 文件，选择“Web Services”→“Generate Client”，如图 3-369 所示。

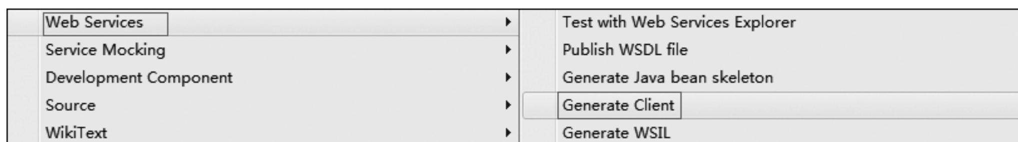


图 3-369 创建客户端的菜单选择

9）客户端的配置级别选择“Develop Client”开发级别客户端，随后按默认配置单击“Next”按钮，如图 3-370 所示。

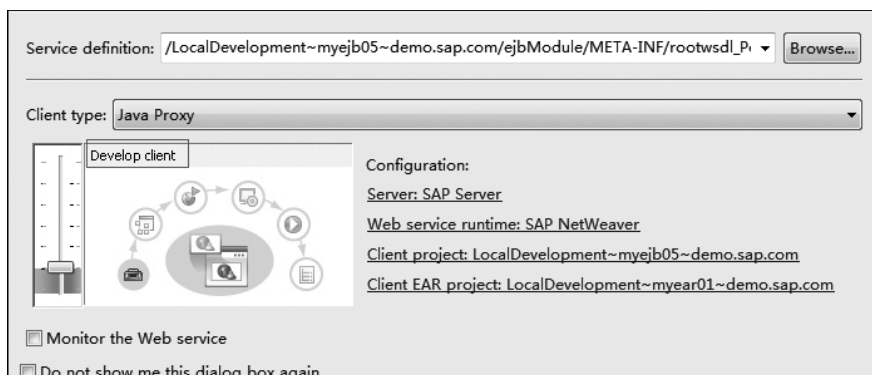


图 3-370 配置服务级别

10）客户端生成后，就能在对应的包中看到自动生成的代理类，如图 3-371 所示。

11）调用对应的 Web 服务，新建 Session Bean，并填写相关的包名和类名，如图 3-372 所示。

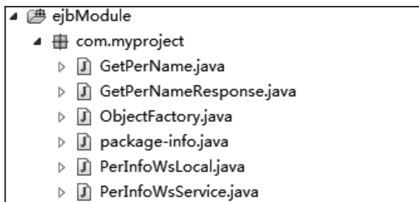


图 3-371　自动生成的类文件

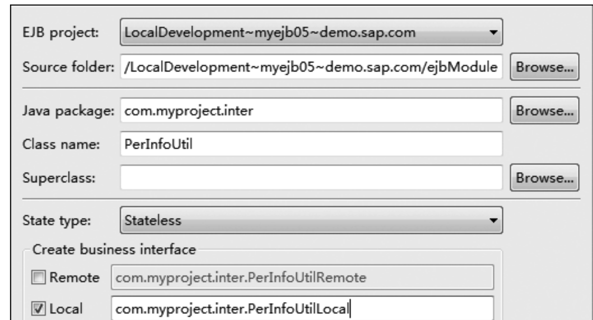


图 3-372　输入项目信息

12）接口中声明获取用户名的“getPerName”方法，代码示例如下：

```
public interface PerInfoUtilLocal {
    public java.lang.String getPerName(String perId);
}
```

13）切换到 Session Bean，定义导入的 Service 引用，通过获取 ServiceInterface 实现该方法，代码示例如下：

```
package com.myproject.inter;

import javax.ejb.Stateless;
import javax.xml.ws.WebServiceRef;

import com.myproject.PerInfoWsLocal;
import com.myproject.PerInfoWsService;

/**
 * Session Bean implementation class PerInfoUtil
 */
@Stateless
public class PerInfoUtil implements PerInfoUtilLocal {
  @WebServiceRef(name = "PerInfoWsService")
  PerInfoWsService service;

  /**
   * Default constructor.
   */
public PerInfoUtil() {
    // TODO Auto-generated constructor stub
}
@Override
public String getPerName(String perId) {
    // TODO Auto-generated method stub
    PerInfoWsLocal perInfoWsPort = service.getPerInfoWsPort();
    return perInfoWsPort.getPerName(perId);
}
}
```

14）保存所有修改即完成了 Web 服务调用的 EJB 实例，其他工程通过直接调用该 EJB 来访问创建的 Web 服务，或者可以将 EJB 再次发布 Web Service 并对客户端程序做测试。

4. 消费端服务配置

新建的 EJB 工程通过 EAR 项目部署到 SAP Java 应用服务器后，能够在 NWA 中消费端服务功能中查找该客户端代理。

1）使用管理员账号登录 NWA 平台，单击“Configuration”→“Connectivity”命令，单击“Single Service Administration”链接并找到“Consumer Proxies”消费者代理配置，如图 3-373 所示。

图 3-373　查看服务列表

2）在“Configuration”配置页签中可以对 Web 服务进行详细配置，如图 3-374 所示。

图 3-374　代理的逻辑端口

3）配置明细包含代理类调用的 Web Service 地址以及 Web 服务的安全认证等，如图 3-375 所示。

图 3-375　逻辑端口的详细配置

4）单击“Edit”编辑“Security”安全信息，输入访问 Web Service 时输入的用户认证信息，如图 3-376 所示。

在 Web 服务接口保持不变，而 WSDL 地址发生变更时，Consumer Service 是对应于 Java

Client 的服务端配置，通过后台修改这些参数可以灵活地应对接口服务端服务地址或认证信息的变动，并不需要做任何设计时的修改。

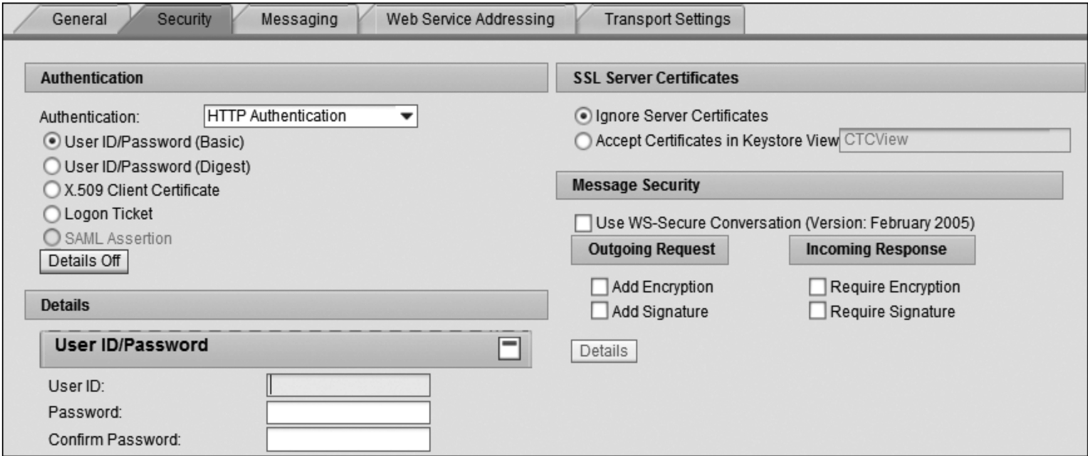


图 3-376　安全配置

3.7.4　EJB 模型实例

与通常的 JavaBean 不同，EJB 具备可分布计算的特性。EJB 发布到服务器后可以通过 JNDI 访问对应的 EJB 服务。

由于之前在创建 Web Service 时已经建立了名为 PerInfoWs 的 EJB，这里不再重复创建的步骤，直接引用该 EJB，如图 3-377 所示。

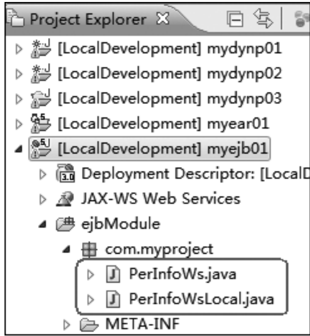


图 3-377　原 EJB 项目示例

开发实例：

1）使用现有的 Web Dynpro 工程，或重新创建一个简单工程。本例以之前创建的 mydynp03 为例，如图 3-378 所示。

2）右击 “Models”，在弹出的快捷菜单中选择 “Create Model”，如图 3-379 所示。



图 3-378　原 Web Dynpro 项目目录

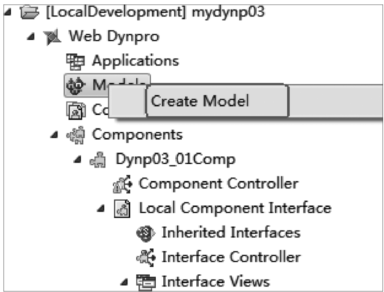


图 3-379　创建模型的菜单

3）在弹出的窗口中选择对应的模型类型，这里选择第三项 “Enterprise JavaBean Model”，单击 “Next” 按钮，如图 3-380 所示。

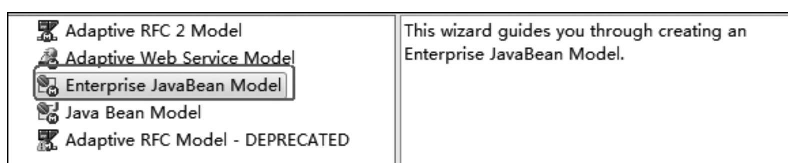


图 3-380　选择模型类型

4）输入导入的模型名称，并且添加依赖 EJB 所在的 DC（DC 类型为 EAR），单击“Next”按钮，如图 3-381 所示。

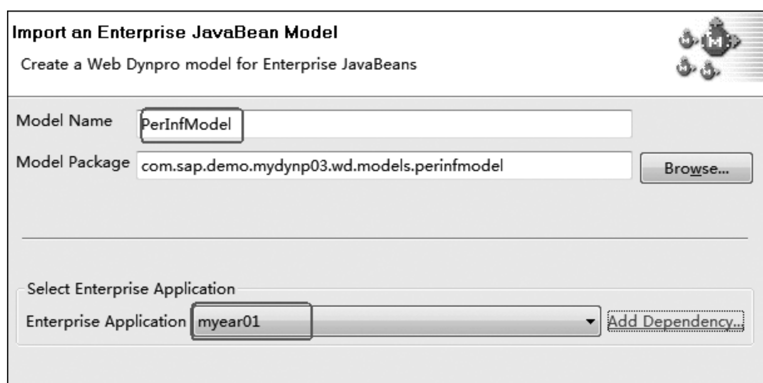


图 3-381　输入模型信息

注：EAR 工程 public parts 需要添加对 EJB 的引用（添加 DC 依赖过程可参考 3.6.3 节）。

5）选中对应需要导入的 EJB，并以默认配置完成后续向导，如图 3-382 所示。



图 3-382　选择方法

6）创建自定义控制器，且完成其上下文与模型的映射，如图 3-383 所示。

7）创建视图及按钮（或直接使用已存在按钮），并在代码编辑中添加对自定义控制器中执行方法的调用，如图 3-384 所示（具体过程请参考 3.6.3 节）。

8）至此，所有配置和代码工作都已完成，保存所有修改并编译部署至服务器，如图 3-385 所示。

9）单击按钮后，就能在页面下方看到 EJB 模型返回的姓名信息，如图 3-386 所示。

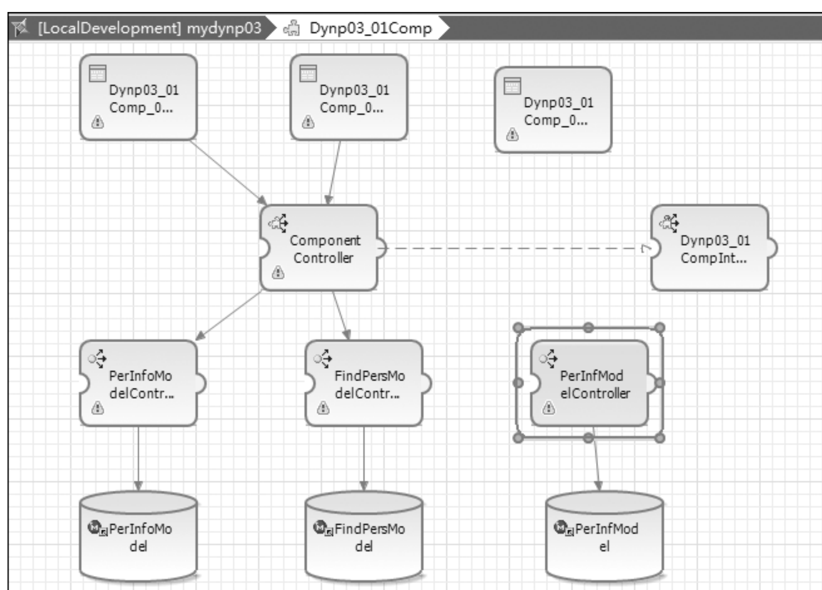


图 3-383　添加上下文映射后的效果

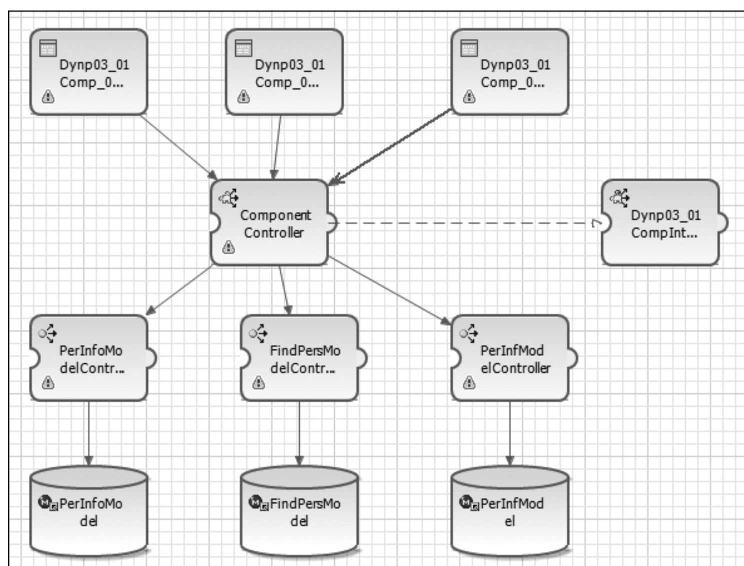


图 3-384　添加方法调用后效果

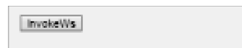


图 3-385　运行效果

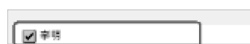


图 3-386　单击按钮后的效果

3.7.5　应用场景贴士

1. 公共模型

根据项目的需求，接口的数量是有限的，但往往多个功能都会访问同一个接口。例如，通过登录 ID 从 ECC 端获取该人员的 HR 人员信息，这属于一个公共接口，所有的子应用在初始化时都会通过调用该接口来显示登录的用户信息。这样就会导致每一个功能开发人员都

要创建针对同一接口的客户端代理（Model），带来了较多不必要的开发量，同时导致后期接口结构变化不易于维护。此时可以通过在公用 DC 中创建公共模型的形式，让所有的开发人员统一使用同一模型去调用接口，这样一方面降低了各自单独创建模型带来的工作量，另一方面即使后期接口发生变化，也只需要修改一个模型，方便了管理和维护。

可能一些人会提到，开发人员是采用了同一个模型，但却分别创建了很多自定义控制器。实际上正如上面提到的，如果该接口确实是公共接口，那就应该在公共 DC 中对接口进行封装，具体的方法就是在公共 DC 的组件中创建自定义控制器以及在它的组件控制器中封装调用接口的方法，并最终通过接口控制器和开发组件的公共部分声明给外部 DC 调用。这样实际上就是进一步地减少了功能开发人员接口调用的开发，省去了创建自定义控制器的过程，同时可以通过直接访问公共 DC 组件的方法来实现对外部系统服务的调用。

2. 接口变更

系统间调用接口时，往往会因为初期双方沟通、调研不足或没有合理预留有效字段造成接口的输入/输出参数不能够满足实际业务需求，这样会导致频繁的接口结构变化，进而影响到客户端接口的访问。实际上，服务端接口结构的变化对客户端接口调用来讲影响是很大的，有时候甚至会导致整个接口调用的重新开发，由此会带来较大的重复工作量。实际的项目开发中应该从前期的接口设计着手，尽可能地让接口满足多样的需求，如充分地了解需求或预留一些备用字段等，同时还应该做好对接口结构变化的应对处理。Web Dynpro 的开发工具提供了“Reimport”重新导入模型的操作来使客户端的代理类与服务端同步，自定义控制器中的上下文也可以通过修改上下文映射使上下文结构与重导入的模型接口结构保持一致。

3.8 应用程序

从之前不同的 Web Dynpro 实例中可以看到，应用程序是最终 Web Dynpro 项目运行的实体，也是服务器端实际挂载和运行的对象。

以之前实例中的应用程序为例，双击打开其中的应用程序，接着来了解一下 Web Dynpro 应用程序的一些具体属性，这其中包括了基本属性（General）、应用程序引用（Application References）和应用程序属性（Application Properties）三个子页签的内容。

1）在基本属性子页签中，看到的是它的一些基本属性，如名称和描述等内容，如图 3-387 所示。

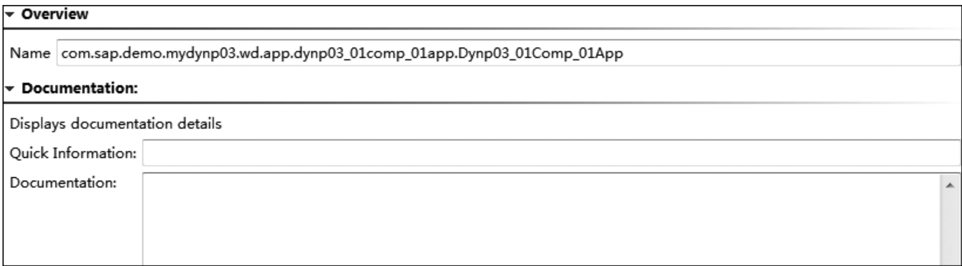


图 3-387 应用程序的基本信息

2）在应用程序引用子页签中可以看到，应用程序在运行时默认显示的实际接口视图内容，此处也可以在创建应用完毕后根据实际需要进行调整和修改，如图 3-388 所示。

References
Displays the details of the launch
Web Dynpro Component Dynp03_01Comp - com.sap.demo.mydynp03.wd.comp.dynp03_01comp
Interface View Dynp03_01Comp_01IntV
Startup Plug Default

图 3-388　应用程序的引用信息

3）在应用程序属性子页签中，可以针对应用程序定义一些特殊的参数，如是否需要权限验证、过期时间等，如图 3-389 所示。

Application Properties
Displays the application properties

Name	Type	Description
sap.authentication	true	Authentication mode

Add

图 3-389　应用的属性

4）如果需要添加参数，则单击“Add”按钮就可以进行修改，如图 3-390 所示。

Type
Predefined
User defined
Name
Browse
Value
Description

图 3-390　添加属性

5）SAP 也提供了许多预定义的应用程序属性，供开发人员进行配置和使用，如图 3-391 所示。

Select Predefined Application Property
Select an application property
Authentication (Authentication mode)
DefaultLocale (Default locale for anonymous users)
DomainRelaxation (Specifies whether domain relaxation is used when r
ExpirationTime (The time in seconds, until a session without user respo
iViewSplitting (Defines whether the application should be split into seve
LogoffURL (URL to a page which is displayed after logging off)
SuspendableApplication (Controls whether the application is suspenda
OK Cancel

图 3-391　选择属性类型

第 4 章　用户界面元素

对于一般的开发人员来说，代码是一个程序最核心的部分，但在实际项目中，不论是产品开发还是二次开发，单元测试、集成测试、压力测试已经在很大程度上完成了代码功能部分的验证，而应用程序的界面（User Interface，UI）才是用户直接接触的部分，所以界面实现的功能、布局设计和易用性更多地决定了实际的最终用户体验。Web Dynpro 提供了丰富的界面元素对页面进行“装潢”，本章将逐一对这些界面元素进行介绍，同时还提供具体的开发实例。

4.1　界面元素介绍

接触过网页开发或其他页面搭建的用户，应该经常会接触到“控件”或者“UI 控件”这样的描述，而在 Web Dynpro 中确切的叫法是“用户界面元素”或“UI 元素”（User Interface Element）。在刚开始接触 Web Dynpro 开发时，可能会觉得界面元素和一般的页面控件非常相似，如按钮、表格、下拉列表等，但其实通过实际的学习和应用，就会发现很多Web Dynpro 的界面元素不只是一个简单的 HTML 控件，它还封装相应的事件并作为接口来提供一些功能。例如，表格类型的元素就默认绑定了 6 个标准事件，如图 4-1 所示。

Table
General
LayoutData
Events
Property | Value
Events
onColSelect
onDrop
onFilter
onLeadSelect
onSelect
onSort

图 4-1　表格元素的标准事件

4.1.1　元素的添加

在 Web Dynpro 中，一种添加 UI 元素的方法是打开视图的布局编辑器，直接从编辑器的左侧将 UI 元素拖曳至视图上，当鼠标悬停在 UI 元素上时，会有对应元素的一个简单介绍弹出，如图 4-2 所示。另一种添加 UI 元素的方法是通过左下角的元素清单列表（Outline 元素大纲）。清单是当前打开视图中所有页面元素一个树状结构的体现，也清晰地展现了各个 UI 元素直接的包含或者并列关系，如图 4-3 所示。

右击任一个容器元素，如右击“RootElement”根元素，弹出菜单的第一项就是“Insert Child”，在弹出的对话框中即可创建不同的 UI 元素，如图 4-4 所示。

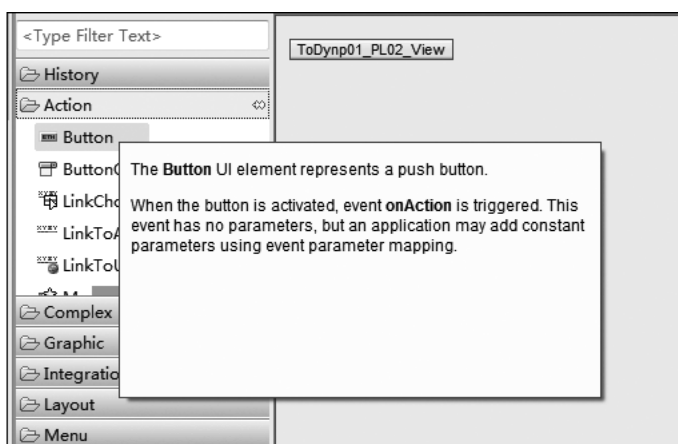


图 4-2　元素的简单介绍

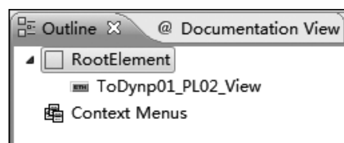


图 4-3　“Outline”视图

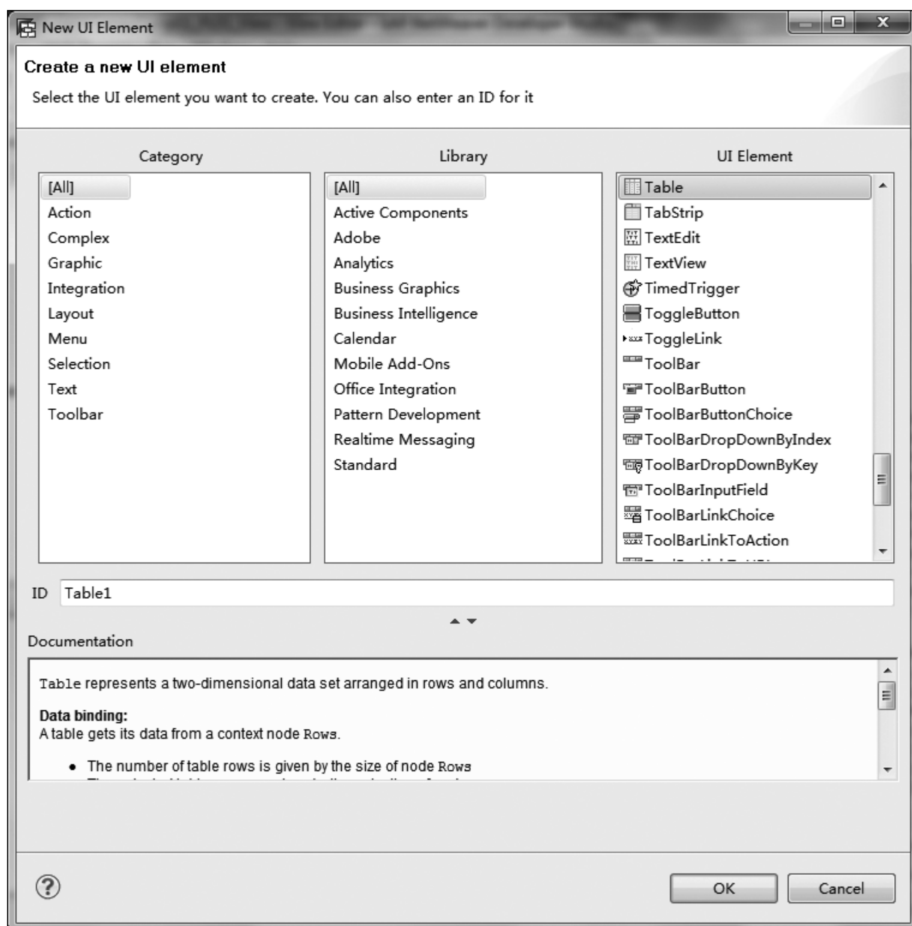


图 4-4　不同的元素类型

在图 4-4 中共有 3 栏，分别表示分类目录、对应类库和具体 UI 元素选择。前两个分类的选择只是为了缩小可选 UI 元素，熟悉各个对象之后直接在最后一栏通过名字或首字母快捷键进行查找即可。中间的“ID”文本框则是用于填写创建 UI 元素的唯一标识，默认情况

下系统对于重复元素会自动进行顺序编号，不会出现重复的标识。最下方的文档说明是对所选择的 UI 元素的定义和简单的使用说明。在选择对应的 UI 元素后，单击“OK”按钮即可完成一个 UI 元素的添加。

在这里一直用“添加”而不是“创建”，是因为对于这种图形化的开发方式，其实真正的创建过程是由开发工具在拖曳元素或单击菜单的过程中自动创建的。

4.1.2　元素的通用属性

这些界面元素对应的 Java 类都是自动生成的，而从逻辑角度来说，所有的界面元素都是以接口的方式被定义。其中，所有的独立元素都是从“IWDUIElement”和“IWDViewElement”两个抽象基础类继承而来，而复合元素则是从“IWDViewElement”抽象类继承而来。所有这些元素都有以下的一些通用属性。

1）标识（id）：用来描述视图元素的唯一标识。

2）是否可用（enabled）：是否运行通过用户交互触发元素对应的事件。

3）提示信息（tooltip）：当用户鼠标悬停在元素上时，弹出气泡文本用于显示简要的说明信息。

4）是否可见（visible）：用于标识视图元素是否显示在页面中。它共有以下 5 个选项。

- always：始终可见，而且用户在进行客户端页面个性化配置时也无法隐藏。
- blank：空白，不显示元素，但占用对应空间。
- none：不可见，且不占用空间。
- notYet：默认是不可见状态，但用户在进行客户端页面个性化配置时可以显示。
- visible：默认可见的状态

5）布局参数（LayoutData）：根据元素所在容器的布局，会有特定的布局参数进行配置。

需要注意的是，部分属性在选中后右侧会出现一个“Bind”按钮，表示该属性允许与上下文中的特性进行绑定，从而更灵活地控制页面元素的显示和使用。还有部分属性在空值时，会有黄色叹号的图标出现作为提醒，这表示该属性不能为空，否则程序无法运行。

4.1.3　元素的布局

在 Web Dynpro 中，布局定义了对应容器中所有 UI 元素的排列和位置，仅有容器类型的元素可以找到对应属性修改布局（如透明容器、组、托盘等）。如果需要修改整个页面的布局，则可以通过修改“RootUIElement”根结点的布局属性来实现，而对于各个子对象的布局设定，则可通过子对象的布局参数进行调整。

假设添加了一个托盘元素，需要注意的是，它所指定的“Layout”布局是针对它的子对象所设定的布局，而左侧依然可以看到的“LayoutData”布局参数则是将托盘作为一个子对象时需要设定的属性，针对的是它的父对象，如图 4-5 所示。这两块属性的配置千万不能混淆。

在 Web Dynpro 中默认可用的布局有 5 种。

1）ColumnLayout（列布局）：强调容器中子对象的横向布局，默认为 3 列，也可以理解为一个沙盘上默认划了 3 道竖线，随后

图 4-5　元素的属性标签

去摆放它的子对象。

2）FlowLayout（流布局）：以一个队列的方式去排列容器中的子对象，按从左到右、从上到下的顺序依次排列，开发人员可以自定义换行的位置，默认根据 UI 元素的宽度以及浏览器窗口的宽度来决定是否换行。

3）GridLayout（网格布局）：所有子对象分布在一个二维的网格之中，网格的行数和列数都可以自由配置。

4）MatrixLayout（矩阵布局）：所有子对象分布在类似于表格的布局中，与网格布局较为相似。它不能像网格布局一样自由配置行数和列数，只能通过将对应的元素设置为"MatrixHeadData"来进行换行。矩阵布局的优势在于，开发人员可以用它的单元类方便地创建出一致布局的结构。

5）RowLayout（行布局）：强调容器中子对象的纵向布局，它其实与矩阵布局类似，但它是将所有子对象顺序地放入一列中。

4.1.4　元素设计规则

为了能在 Web Dynpro 中更好地进行界面设计和展现，介绍一些简单的设计规则：

- 不要错用 Web Dynpro 中的 UI 元素，尽可能了解不同元素的作用。
- 尽量避免使用固定的宽度和高度，同时要考虑应用程序在不同客户端的计算机中可能会使用不同的浏览器或不同的分辨率的情况，需要根据实际情况进行具体调整。
- 在使用透明容器进行多元素嵌套时，需要考虑不同容器的不同布局，尽可能避免不必要的元素嵌套。
- 不要在嵌入容器中使用滚动容器，因为在出现多重的滚动页面时，用户经常会无法正确移动对应的视图。
- 在 Web Dynpro 代码编写时，对于视图初始化的代码，建议放在"wdDoInit"方法中而不是"wdDoModifyView"中，这是为了避免视图修改时重复调用初始化代码，降低处理效率。

4.2　简单元素

4.2.1　按钮

按钮是最常用的元素之一，大部分用户事件都是通过单击按钮进行触发和响应的。

按钮包含以下属性。

- activateAccessKey：设置是否允许快捷键（使用"Alt"键与首字母触发默认事件）。
- contextMenuBehaviour：设置上下文菜单行为，默认有"继承""允许"和"禁止"3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- Design：控制按钮的显示风格，默认有标准显示、增强显示、下一步和上一步 4 种类型，其中增强类型即为文本加粗显示，见表 4-1。
- enabled：控制元素是否可用。
- hotkey：设置指定键盘组合作为快捷键使用，必须从下拉菜单选择不可自定义设置。

表 4-1　按钮的 4 种类型

类　型	图　例	说　明
标准显示	standard	显示标准按钮的样式
增强显示	emphasized	以高亮方式显示按钮的样式
下一步	next ▶	按钮右侧增加了一个向右的箭头，一般用于表示下一步
上一步	◀ previous	按钮左侧增加了一个向左的箭头，一般用于表示上一步

- imageFirst：控制显示的链接图标是否位于文本前面，布尔型属性。
- imageSource：控制按钮图标的图片源，可以选择 sap 标准图标或自定义图片。
- text：控制按钮的显示文本。
- textDirection：控制链接文本的排列顺序，默认有 3 个选项：继承自父元素、从左往右和从右往左。
- tooltip：控制提示消息的显示文本（对应老版本中的“imageAlt”属性）。
- visible：控制元素是否可见。
- width：控制元素的显示宽度。
- onAction 事件：用于绑定按钮单击后的响应动作。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择按钮类型，输入 ID 名称，单击“OK”按钮，如图 4-6 所示。

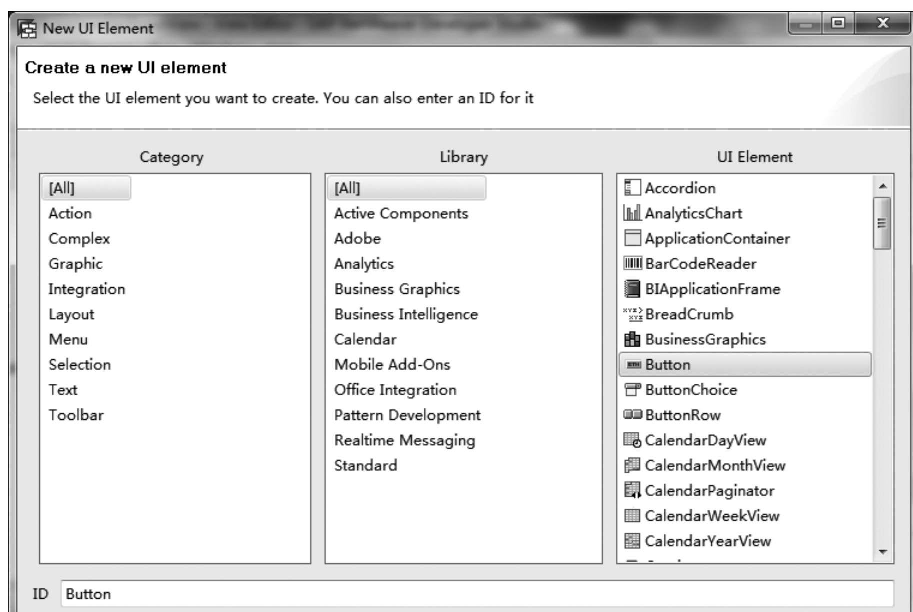


图 4-6　选择元素类型

2）修改按钮属性中的“text”文本，改为“提交”，如图 4-7 所示。

3）修改按钮属性中的“imageSource”图源，选择一个 SAP 标准图标“s_b_okay. gif”，如图 4-8 所示。

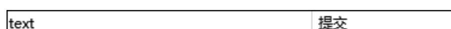


图 4–7　修改文本属性



图 4–8　修改图源属性

4）切换到视图的上下文中，创建一个结点。由于仅用于特性管理，而不是作为表格行项目，因此结点的集合属性设置为“1‥1”，如图 4–9 所示。

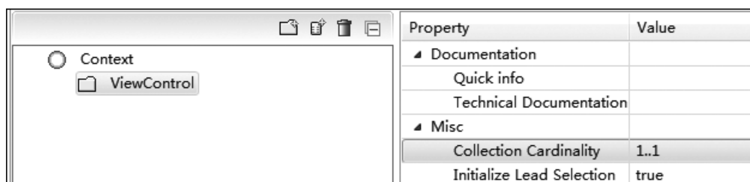


图 4–9　设置结点属性

5）创建类型为“Visibility”的特性“Bt_Visi”。选择创建一个特性，输入特性名称，单击“Browse”按钮进行特性类型的选择，如图 4–10 所示。

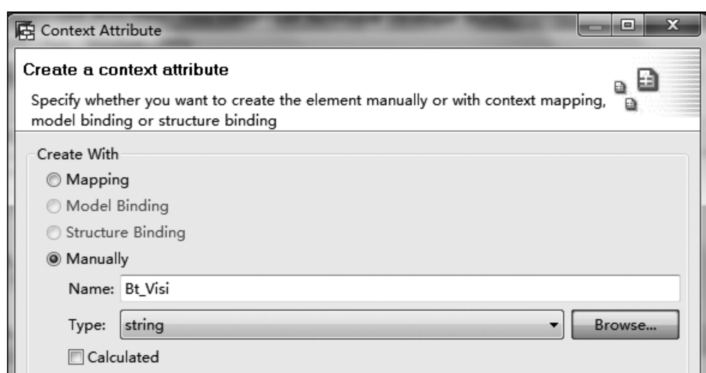


图 4–10　创建特性

6）在弹出对话框中定位到“com. sap. ide. webdynpro. uielementdefinitions”包，展开并找到可见的类型对象“Visibility”，选中并单击“OK”按钮，如图 4–11 所示。

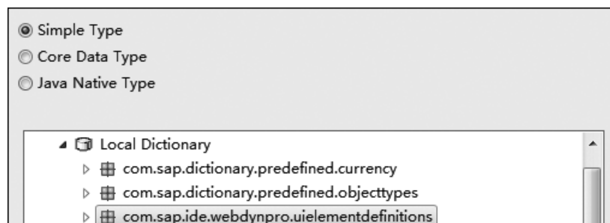


图 4–11　选择复杂类型

7）完成类型选择，确认信息无误后，单击“Finish”按钮，如图 4–12 所示。

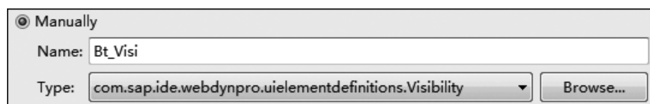


图 4–12　完成类型选择后效果

8）新增“boolean”的特性“Bt_Ena”，如图 4-13 所示。

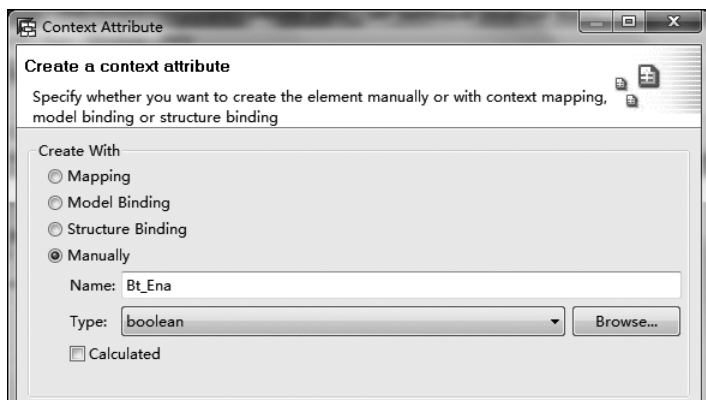


图 4-13　新建特性

9）完成特性创建后，保存所有修改，如图 4-14 所示。

10）单击选中按钮元素，将创建的特性与按钮属性进行绑定，选中对应属性，单击右侧的“Bind”按钮进行特性绑定，如图 4-15 所示。

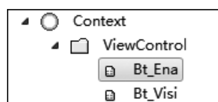


图 4-14　创建完毕后的效果



图 4-15　属性修改

11）至此，完成了按钮一般属性的配置工作，如图 4-16 所示。

12）单击左侧的“Events”事件标签，进行按钮动作定义。单击右侧的“Create”按钮，如图 4-17 所示。

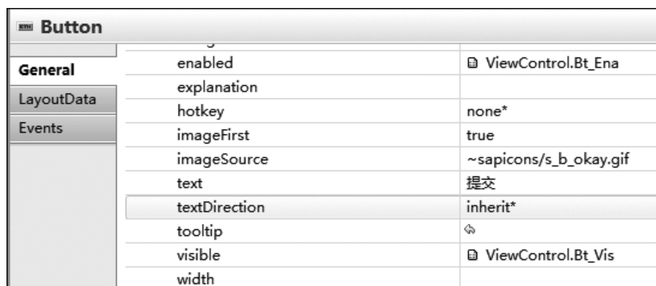


图 4-16　配置后的效果

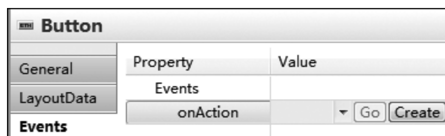


图 4-17　创建事件对应动作

13）在弹出的对话框中输入动作名称，单击“Finish”按钮，如图 4-18 所示。

14）保存所做修改后，双击“Submit”方法或切换到 Java 编辑器，实现事件处理器。添加代码，如图 4-19 所示。

```
wdComponentAPI. getMessageManager( ). reportSuccess( "Submit" ) ;
```

15）创建组件的应用程序后，保存所有修改，并重新编译、运行之后，即可进行相关测试。此时按钮为灰色，如图 4-20 所示无法使用。这是因为我们将是否可用的属性与视图

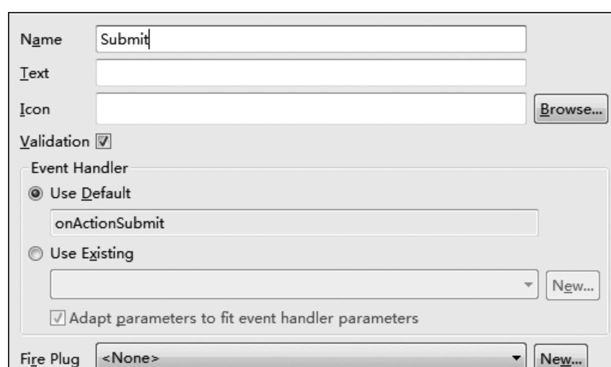


图 4-18　输入动作名称

```
/**
 * Declared validating event handler.
 *
 * @param wdEvent generic event object provided by framework
 */
//@@end
public void onActionSubmit(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionSubmit(ServerEvent)
    wdComponentAPI.getMessageManager().reportSuccess("Submit");
  //@@end
}
```

图 4-19　添加代码

上下文进行了绑定，但未对上下文进行赋值，默认为空值，即对应“false”的显示效果。



图 4-20　运行效果

16）进入到视图的“wdDoInit”初始化方法中，初始化时赋值该变量为“true”，如图 4-21 所示。

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit()
{
  //@@begin wdDoInit()
    wdContext.nodeViewControl().currentViewControlElement().setBt_Ena(true);
  //@@end
}
```

图 4-21　添加代码

17）重新部署运行后，“提交”按钮可用，单击该按钮即可看到对应的提示消息，如图 4-22 所示。



图 4-22　运行效果

18）对应另一个是否可见的上下文，也可以同样在初始化方法中进行赋值，如图 4-23 所示。

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit()
{
  //@@begin wdDoInit()
    wdContext.nodeViewControl().currentViewControlElement().setBt_Ena(true);
    wdContext.nodeViewControl().currentViewControlElement().setBt_Visi(WDVisibility.NONE);
  //@@end
}
```

图 4-23　添加代码

编译运行后，提交按钮即不再显示，读者也可以根据通用属性中的介绍，进行其他的可视属性配置，并验证实际页面中的显示效果。

4.2.2 文本框

文本框包含以下属性。

- activateAccessKey：设置是否允许使用快捷键（使用“Alt”键与首字母触发默认事件）。
- alignment：控制文本框中文本显示的位置，如靠左、靠右或居中等。
- autoTab：控制用户是否可以通过“Tab”键进行快速跳转，布尔型属性。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- defaultDateRefId：设置默认的引用日期标识。
- displayAsText：控制是否以文本形式显示，如设置为“true”，则在视图中的显示效果与“TextView”一样。
- enabled：控制元素是否可用。
- imeMode：控制是否允许本地计算机中的输入法进行文本输入。
- inputPromt：控制焦点在文本框中时的默认输入提示。
- length：控制文本框的显示长度。
- onEnterBehavior：控制在文本框中回车时的响应事件。
- passwordField：控制是否以密码方式显示输入值，即当属性值为 true 时，输入框将显示“＊＊＊”作为密码输入。
- readOnly：控制文本框是否只读。
- state：控制元素的显示状态，默认有“normal”和“required”两种状态。在选择“required”后，元素会出现一个红色星号进行提醒，但逻辑控制仍需要编写相关代码。
- suggestedValue：控制是否有推荐值。
- textDirection：控制链接文本的排列顺序，默认有 3 个选项：继承自父元素、从左往右和从右往左。
- tooltip：控制提示消息的显示文本。
- value：文本框中输入值所对应的上下文特性，文本框的关键属性。另外，文本框元素的显示方式与绑定变量类型有直接关系，当变量类型为日期类型时，文本框元素会自动显示日期搜索帮助。同样当类型为数字或整型时，界面元素会提供数字型搜索并有类型合法性校验。
- visible：控制元素是否可见。
- width：控制元素的显示宽度。
- onChange 事件：当文本框的值发生改变并且失去鼠标焦点或按〈Enter〉键时触发。
- onEnter 事件：当按〈Enter〉键时触发（如果文本框属性是“SuggestValues”并且“onEnterBehavior”设置为“no”时，则不会触发该动作）。

注：当存在“onChange”事件时，“onEnter”事件不会被触发，即不论是文本框的值发

生改变、输入框失去鼠标焦点，还是按〈Enter〉键，都会优先触发“onChange”事件，而跳过“onEnter”事件。

开发实例：

1）打开任意一个视图，添加一个文本框。右击“Outline”中的“RootElement”根结点，选择插入子对象，选择“InputField”，输入 ID 名称，随后单击“OK”按钮，如图 4-24 所示。

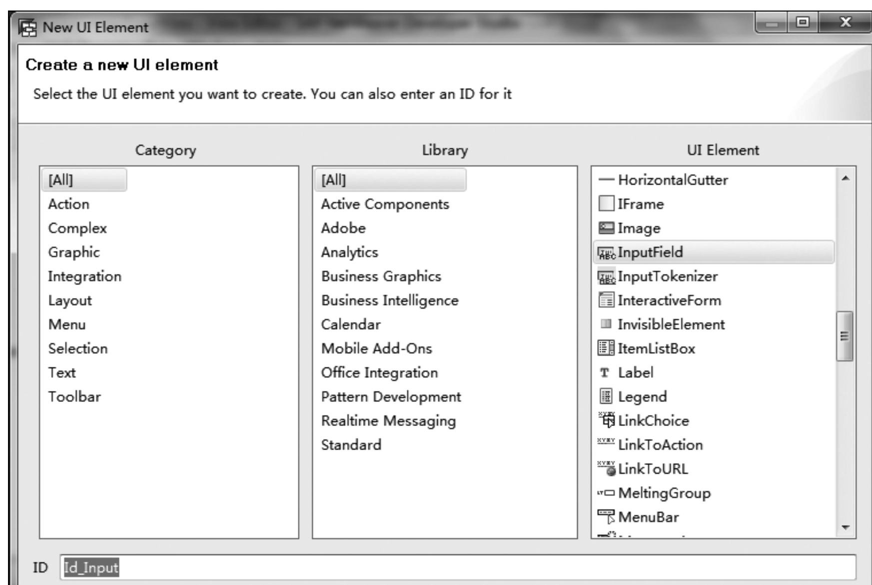


图 4-24　选择元素类型

2）再添加一个标签，右击“Outline”中的“RootElement”根结点，选择插入子对象，选择“Label”，输入 ID 名称，随后单击“OK”按钮，如图 4-25 所示。

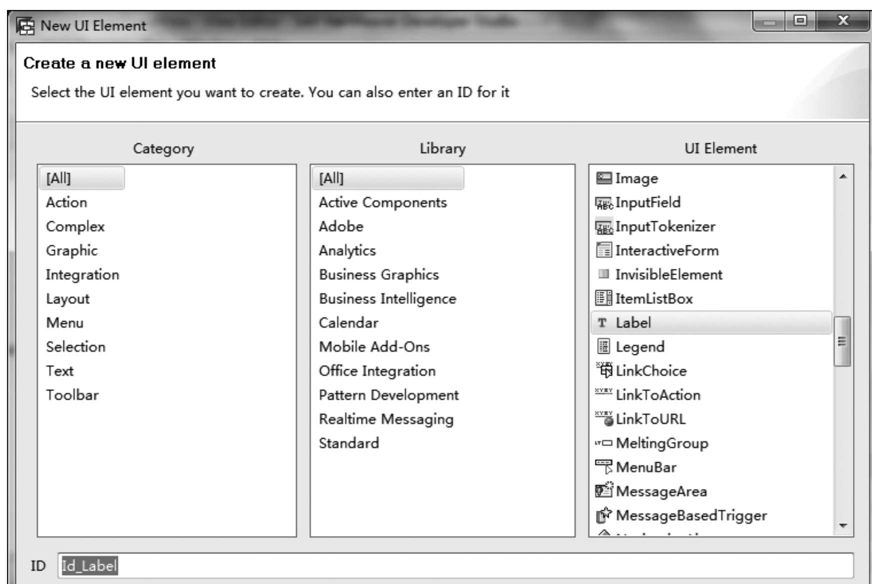


图 4-25　选择元素类型

3）打开标签的属性，在“labelFor”属性中选择之前创建的“Id_Input”的文本框的ID，“text”属性输入“Id”，如图 4-26 所示。

4）完成标签的修改后，接着打开文本框的属性，在“state”状态属性中选择“requied”表示文本框必填，如图 4-27 所示。

Label

Property	Value
◢ Properties [Label]	
id	Id_Label
contextMenuBehaviour	inherit*
contextMenuId	
design	standard*
designBar	standard*
enabled	true
labelFor	Id_Input
text	Id

图 4-26　修改标签属性

InputField

onEnterBehavior	default*
passwordField	false
readOnly	false
state	required

图 4-27　修改输入框属性

5）完成修改后，保存所有修改，在视图中会看到文本框前多了一个红色星号，提示用户必填，但元素中属性修改只是页面上提示，逻辑判断还需要额外编写代码实现，如图 4-28 所示。

6）在“inputPromt”属性输入“Enter Id”，保存修改后，页面效果如图 4-29 所示。

7）如果这时保存并编译、运行程序，会提示文本框的值未绑定上下文，这是由于文本框对于用户的输入必须有对应的上下文进行保存。切换到视图的上下文中，创建一个名称为“Id”、类型为“String”的特性，如图 4-30 所示。

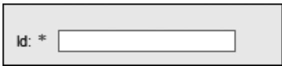


图 4-28　预览效果

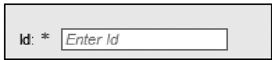


图 4-29　运行效果



图 4-30　创建特性后的效果

8）将文本框的属性中的“value”与上下文进行绑定，如图 4-31 所示。

9）保存所有修改后，编译、部署并运行视图对应的程序，运行效果如图 4-32 所示。

InputField

onEnterBehavior	default*
passwordField	false
readOnly	false
state	required
suggestValues	false
textDirection	inherit*
tooltip	
value	Id
visible	visible*

图 4-31　修改属性后的效果

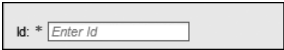


图 4-32　运行效果

10）下面继续测试文本框的其他属性以及实现后的实际效果。首先，在文本框后添加一个按钮，用于后面的测试。在“text”文本框中输入“打印”，图标选择“SAP Icon”标准图标中的“s_b_prnt. gif”，如图 4-33 所示。

11）随后添加一个方法，在代码中添加打印逻辑，用于打印文本框内的文本，如图 4-34 所示。

```
wdComponentAPI. getMessageManager( ). reportSuccess( wdContext. currentContextElement( ). getId( ) );
```

Button

General	enabled	true
LayoutData	explanation	
Events	hotkey	none*
	imageFirst	true
	imageSource	~sapicons/s_b_prnt.gif
	text	打印

图 4-33　修改属性

12）将文本框的“passwordField”属性设置为“true”，编译运行后的效果如图 4-35 所示。

13）单击“打印”按钮后，下方消息区域会提示输入内容，如图 4-36 所示。

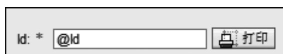


图 4-34　预览效果

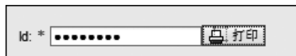


图 4-35　运行效果

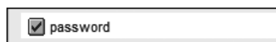


图 4-36　单击打印按钮效果

14）将文本框的“readOnly”只读属性设置为“true”，如图 4-37 所示。

15）在文本框的事件属性中，创建“onChange”和“onEnter”事件，如图 4-38 所示。

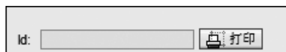


图 4-37　运行效果

Events	
onChange	ChangeValue
onEnter	Confirm

图 4-38　创建事件

“onChange”事件参考代码如下：

```
//@@begin onActionChangeValue(ServerEvent)
wdComponentAPI. getMessageManager(). reportSuccess("OnChange" +wdContext. currentContext-El-
ement(). getId());
//@@end
```

“onEnter”事件参考代码如下：

```
//@@begin onActionConfirm (ServerEvent)
wdComponentAPI. getMessageManager(). reportSuccess("OnEnter" + wdContext. currentContext-
Element(). getId());
//@@end
```

16）保存所有修改，并重新编译，运行即可进行测试，运行效果如图 4-39 所示。

17）经测试可以发现，不论是输入任意文本按〈Enter〉键，还是输入文本后单击页面其他位置，触发的都是“onChange”对应的事件，如图 4-40 所示。

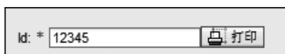


图 4-39　运行效果

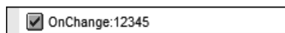


图 4-40　触发事件效果

18）回到视图的属性，将文本框事件中的“onChange”动作清空，仅保留回车事件，如图 4-41 所示。

Events	
onChange	
onEnter	Confirm

图 4-41　修改事件属性

19）保存所有修改，并重新编译、运行之后，输入值并按〈Enter〉键即可触发对应的回车响应事件，如图 4-42 所示。

注：在项目实施过程中，一般回车事件使用的更多，如报表查询时用户经常提出输入有效值后回车即能看到结果，这时候开发人员就需要创建回车事件并且绑定与单击“查询”按钮相同的动作进行响应。

20）在 Web Dynpro 中，文本框也会随着所绑定的上下文类型进行一些特别的显示。例如，将上下文“Id”的类型从“String”字符串改为“date”日期，随后保存所有修改，并重新编、译运行，如图 4-43 所示。

21）文本框的右侧多了一个日历的小图标，单击即可弹出对应的日历控件进行日期选择，如图 4-44 所示。



图 4-42　运行效果 1



图 4-43　运行效果 2



图 4-44　日历控件效果

4.2.3　关键字下拉列表

一个关键字下拉列表给用户提供了一个下拉清单，而且用户仅能选择最多一个条目。这个元素其实是由一个文本域、一个按钮和一个选择列表组成的，单击下拉按钮可以看到清单中所有可选的条目，而选中的条目将会在文本域中进行显示。

关键字下拉列表一直是较常见的页面元素，它的可选范围可以通过字段所对应的类型来个性化。当下拉列表框的条目达到一定数量时，下拉列表元素会自动转换为弹出式的输入帮助对话框。

下拉列表包含以下属性。

- activateAccessKey：设置是否允许快捷键（使用“Alt”键与首字母触发默认事件）。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- displayAsText：控制是否以文本形式显示，如设置为“true”，则在视图中的显示效果与“TextView”一样。
- enabled：控制元素是否可用。
- labelFor：用于设置元素对应的显示标签。
- readOnly：控制元素是否只读。
- selectedKey：绑定控件操作的上下文特性。“selectedKey”为值选择后赋值变量。对于

静态下拉框来讲，下拉框的选择范围可以通过在数据字典中建立简单类型进行指定，此时上下文特性的类型需要选择该简单类型。

- state：控制元素的显示状态，默认有“normal”和“required”两种状态。在选择“required”后，元素会出现一个红色星号进行提醒，但逻辑控制仍需要编写相关代码。
- suggestedValue：控制是否有推荐值。
- textDirection：控制下拉列表显示文本的排列顺序，默认有3个选项：继承自父元素、从左往右和从右往左。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- width：控制元素的显示宽度。
- onSelect 事件：当用户选择某一确定值时，会触发该事件。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择关键字下拉列表类型，输入 ID 名称，单击“OK”按钮，如图 4-45 所示。

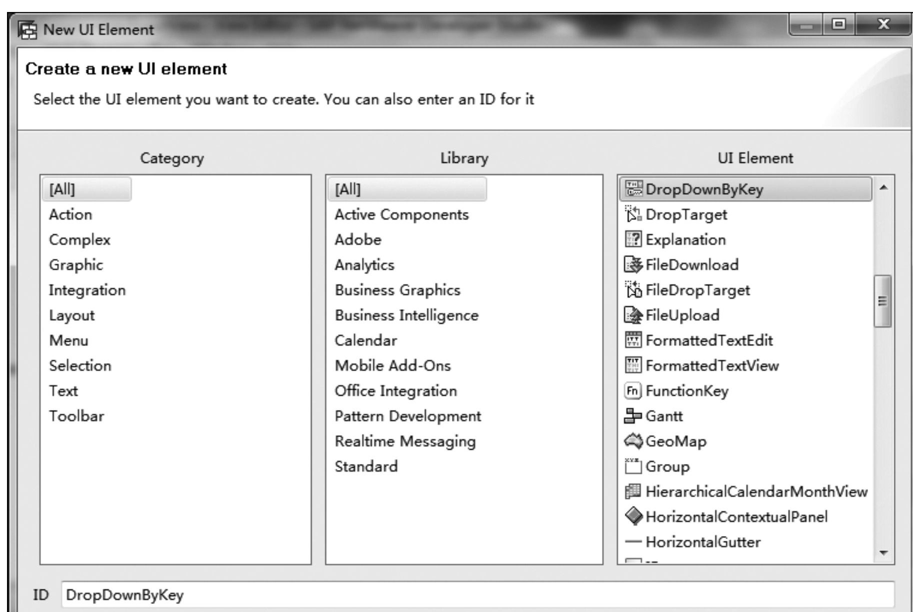


图 4-45　选择元素类型

2）切换到视图的上下文页签中，创建类型为“string”的特性“dept”，如图 4-46 所示。

3）选中关键字下拉列表，将“selectedKey”所选关键字属性绑定为上下文中的“dept”，如图 4-47 所示。

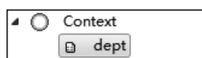


图 4-46　创建特性



图 4-47　属性修改

4）添加一个“Label”标签，将属性中的“text”修改为“部门”。保存所有修改，并编译、运行，运行效果如图 4-48 所示。

5）由于当前绑定的上下文是一个字符串类型，因此并不能实现多个条目的效果。下面创建一个简单类型进行条目填充。在工程目录中（窗口的下方）找到“Ditionaries”→“Local Dictionary”→“Data Types”，右击“Simple Types”，选择“Create Simple Type”，如图 4-49 所示。

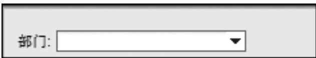


图 4-48　运行效果

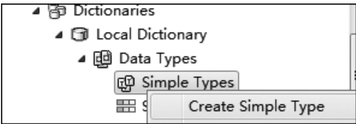


图 4-49　创建简单类型

6）根据需求输入简单类型的名称和包名，其中简单类型的包名也可直接选择现有的包，单击“Finish”按钮，如图 4-50 所示。

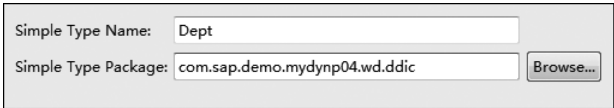


图 4-50　输入简单类型信息

7）创建完成后，即可对创建的简单类型进行详细的配置。上方是一般信息定义，主要关注类型的定义；下方是对于长度常量的定义；最下方还有其他 3 个页签，其中“Enumeration”是定义集合中的所有条目，如图 4-51 所示。

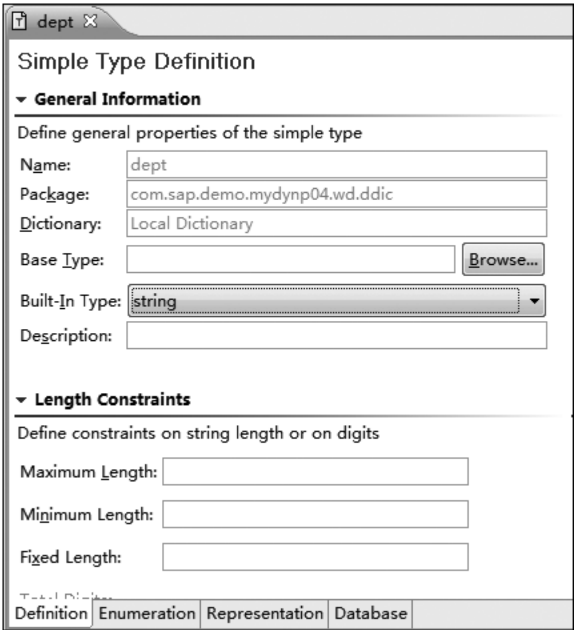


图 4-51　简单类型的定义页面

8）切换到“Enumeration”页签，单击“New”按钮，定义值范围，如图 4-52 所示。

Enumeration Value: 01
Enumeration Text: 人事部

图 4-52　输入信息

9）依次输入多个部门，单击“完成”按钮并保存所有修改，如图 4-53 所示。

Value	Description
01	人事部
02	网络部
03	采购部

图 4-53　完成后的效果

10）回到视图的上下文页签中，选择“dept”特性，并修改它的类型，如图 4-54 所示。注意，修改类型时如果单击下拉列表框只能看到一些基本类型，需要单击右侧的“...”按钮。

11）在弹出的对话框中找到之前创建的包以其对应的“dept”，选中后单击“OK”按钮，如图 4-55 所示。

Property	Value
Calculation	
Calculated	false
Misc	
Name	dept
Read-Only	false
Semantic ID	
Structure Element	
Type	string ... Go

More types...

图 4-54　修改类型

Type Selection
Type Selection
Select a type
Simple Type
Core Data Type
Java Native Type
Dictionaries
Local Dictionary
com.sap.dictionary.predefined.currency
com.sap.dictionary.predefined.objecttypes
com.sap.demo.mydynp04.wd.ddic
dept

图 4-55　选择对应类型

12）保存所有修改，并重新编译、运行之后，就能正确地使用关键字下拉列表了，如图 4-56 所示。

13）细心的读者会发现，下拉列表的宽度自动进行了调整，它会根据所绑定类型中的文本长度自动进行调整，如图 4-57 所示。可以通过修改下拉列表的“width”属性来固定修改元素的宽度。通常在页面布局时并不建议直接将元素宽度设置为固定值，设定百分比的形式会让页面布局变得灵活。

14）保存所有修改，并重新编译、运行之后，就能看到下拉列表最终的运行效果，如图 4-58 所示。

部门:

图 4-56　运行效果

width　100px

图 4-57　“width”属性

部门:
人事部
网络部
采购部

图 4-58　运行效果

15）由于下拉列表中的条目与元素所绑定的简单类型中的“Enumeration”对应，因此再次修改“dept”，添加一行空值，如图 4-59 所示。

Value	Description
01	人事部
02	网络部
03	采购部

图 4-59　修改值

16）保存所有修改，并重新编译、运行之后，选择范围中多出了空值选择，可以通过空值来重置下拉输入，如图 4-60 所示。

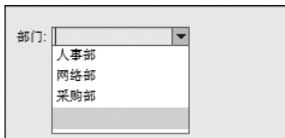


图 4-60　运行效果

17）在下拉列表的默认事件中创建一个动作，如图 4-61 所示。

◢ Events	
onSelect	SelectValue

图 4-61　创建对应事件

18）添加打印代码，即在选择条目时将选中后的上下文的值打印在页面上，如图 4-62 所示。

```
wdComponentAPI. getMessageManager( ). reportSuccess( wdContext. currentContextElement( ). getDept);
```

```
public void onActionSelectValue(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionSelectValue(ServerEvent)
    wdComponentAPI.getMessageManager().reportSuccess(wdContext.currentContextElement().getDept());
  //@@end
}
```

图 4-62　添加代码

19）保存所有修改，并重新编译、运行之后，选择任意一个条目（见图 4-63），页面会打印出关键字（见图 4-64），而不是关键字对应的描述。

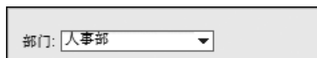


图 4-63　运行效果

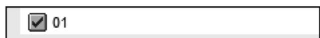


图 4-64　选择后的效果

4.2.4　复选框

复选框允许用户针对某些变量进行真与假（是与否）的选择，如果复选框是选中的状态，则对应的变量值是“true”；如果是未选中状态，则表示的是“false”。

复选框包含以下属性。

- activateAccessKey：设置是否允许使用快捷键（使用“Alt”键与首字母触发默认事件）。
- checked：用于控制复选框是否选中，必须绑定上下文特性。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- enabled：控制元素是否可用。

- explanation：设置元素的解释文本，设置后元素文本下方会出现一条绿色的下画线。当鼠标悬停在文本上时，会弹出解释文本的说明。
- readOnly：控制元素是否只读。
- state：控制元素的显示状态，默认有“normal”和“required”两种状态。在选择“required”后，元素会出现一个红色星号进行提醒，但逻辑控制仍需要编写相关代码。
- text：控制复选框显示的描述文本内容。
- textDirection：控制下拉列表显示文本的排列顺序，默认有 3 个选项：继承自父元素、从左往右和从右往左。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- onToggle 事件：用户选择不同复选框时触发的事件。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择复选框类型，输入 ID 名称，随后单击“OK”按钮，如图 4-65 所示。

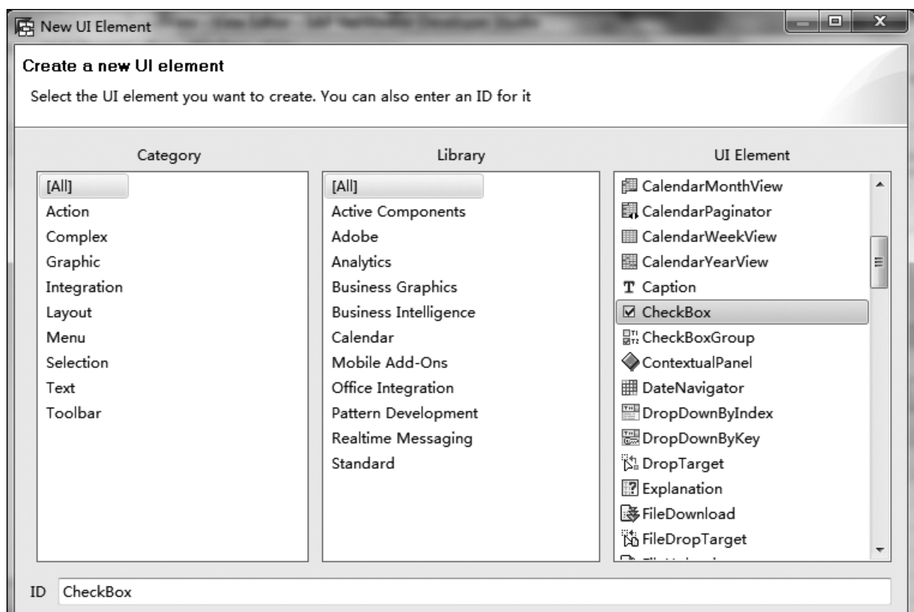


图 4-65　选择元素类型

2）复选框与文本框一样，也有必须与上下文绑定的属性，在属性中可以看到提醒绑定的图片，如图 4-66 所示。

3）创建一个新的特性并与复选框绑定，将“text”文本修改为“按钮可用”，如图 4-67 所示。

4）将同一个特性与按钮的“Enabled”是否可用属性绑定，如图 4-68 所示。

5）保存所有修改，并重新编译、运行之后，会发现此时单击复选框并不会改变按钮的状态，如图 4-69 所示。

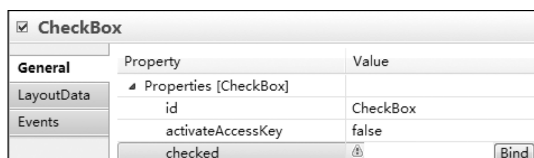


图 4-66　修改属性 1

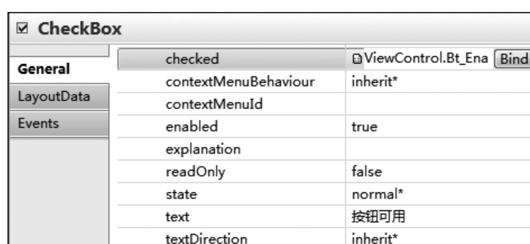


图 4-67　修改属性 2

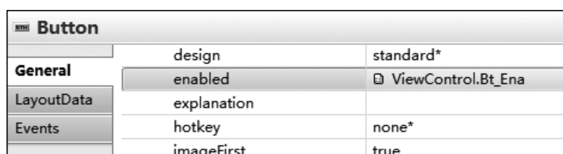


图 4-68　修改属性 3

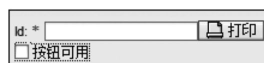


图 4-69　运行效果

6）回到复选框的事件属性中，任意绑定一个动作，即使实现代码为空也没有关系（见图 4-70），保存所有修改，并重新编译、运行。

7）此时单击复选框后，按钮的是否可用属性也会同时修改，未选中的状态时按钮也是灰色不可用状态（见图 4-71）。

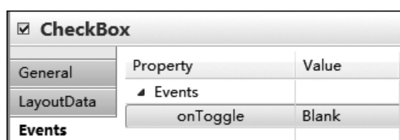


图 4-70　创建事件

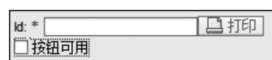


图 4-71　运行效果

在上述测试结果后可以发现，只有在视图元素绑定了特定的事件之后，才会触发视图控制器的逻辑，同时会去刷新上下文特性的值。相对而言，在复选框未绑定事件时，不管用户如何点击复选框，这些操作也仅仅存在于用户客户端的浏览器中，并未触发至服务端进行响应。

4.2.5　单选按钮

单选按钮可以看作一个拥有两个状态（开/关）的按钮，一般也会使用单选按钮组为用户提供选项。当然，如果条目太多，还是建议使用下拉列表。

单选按钮包含以下属性。

- activateAccessKey：设置是否允许快捷键（使用“Alt”键与首字母触发默认事件）。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- enabled：控制元素是否可用。
- explanation：设置元素的解释文本，设置后元素文本下方会出现一条绿色的下画线。当鼠标悬停在文本上时，会弹出解释文本的说明。
- keyToSelect：当前单选按钮对应的键。
- readOnly：控制元素是否只读。

- selectedKey：绑定的上下文结点数据源。
- state：控制元素的显示状态，默认有“normal”和“required”两种状态。在选择“required”后，元素会出现一个红色星号进行提醒，但逻辑控制仍需要编写相关代码。
- text：控制复选框显示的描述文本内容。
- textDirection：控制下拉列表显示文本的排列顺序，默认有 3 个选项：继承自父元素、从左往右和从右往左。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- onSelect 事件：在用户选择时触发的事件。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择单选按钮类型，输入 ID 名称，单击“OK”按钮，如图 4-72 所示。

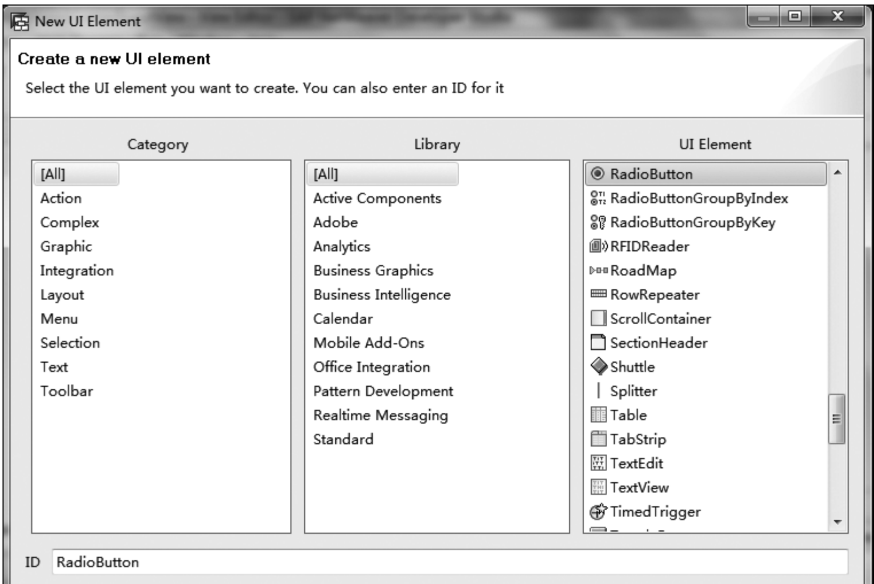


图 4-72　选择元素类型

2）再添加一个单选按钮和一个标签，如图 4-73 所示。

3）编辑标签文本为“性别”，编辑两个单选按钮的文本分别为“男”和“女”，如图 4-74 所示。

4）进入视图的上下文页签中，创建一个字符串类型的上下文“gendor”，如图 4-75 所示。

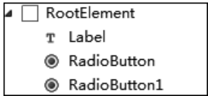


图 4-73　创建后的效果



图 4-74　运行效果



图 4-75　创建特性

5）将第一个单选按钮的“keyToSelect”改为“1”，随后绑定“gendor”上下文特性，如图 4-76 所示。

```
RadioButton
```

6）将第二个单选按钮的“keyToSelect”改为“2”，随后绑定“gendor”上下文特性，如图 4-77 所示。

keyToSelect	1
readOnly	false
selectedKey	gendor

图 4-76　修改属性

keyToSelect	2
readOnly	false
selectedKey	gendor

图 4-77　修改属性

7）在单选按钮的事件中创建“Select”动作，同时将另一个单选按钮的时间处理也与“Select”动作进行绑定，如图 4-78 所示。

8）在创建动作时，添加一个字符串类型的输入参数“key”，如图 4-79 所示。

Events	
onSelect	Select

图 4-78　创建事件

Details | Parameters

Name	Type
key	string

图 4-79　添加动作参数

9）右击“Outline”清单中的单选按钮，选择“Parameter Mapping”创建参数映射（如果在创建动作时不添加对应参数，则无法进行映射），将单选按钮所选的关键字与创建的动作中的参数进行映射，如图 4-80 所示。

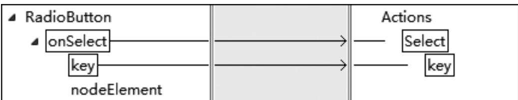


图 4-80　添加参数映射

10）保存所做修改，切换至 Java 编辑器，在“Select”动作的方法中添加代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess( key) ;
```

11）保存所有修改，并重新编译、运行之后，即可测试单击单选按钮的效果，如图 4-81 所示。

12）单击按钮后，可以看到打印出的具体内容，如图 4-82 所示。

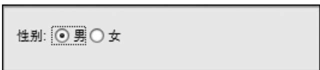


图 4-81　运行效果



图 4-82　选择后的效果

4.2.6　工具栏

工具栏是一个工具的集合，或者可以看成一种特殊的容器，因此工具栏中也可以添加若干子元素用于实现不同的功能，如按钮和下拉列表等，如图 4-83 所示。

工具栏包含了以下属性。

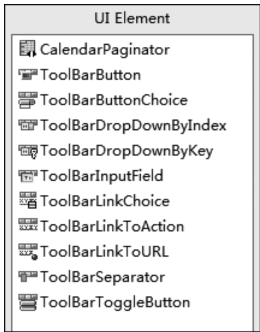


图 4-83　工具栏的子元素

- accessibilityDescription：当这个属性激活时，描述的文本就会显示在提示消息中。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- design：用于指定工具栏的设计样式，默认有“emphasized”和“standard”两种样式。
- enabled：可绑定上下文的属性。控制元素是否可用。
- itemOverFlow：控制是否允许工具栏中的对象超过边界。
- tooltip：控制提示消息的显示文本。
- visible：可绑定上下文的属性。控制元素是否可见。
- wrapping：用于说明工具栏中的元素是否会自动折叠。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择工具栏类型，输入 ID 名称，单击“OK”按钮，如图 4-84 所示。

2）需要注意的是，工具栏也只是个容器，与透明容器、组一样，如果不添加子元素，只会是页面上的一个空壳。右击“ToolBar”工具栏即可添加工具栏支持的子元素，如图 4-85 所示。其中，第一个选项是添加工具栏子元素；第二个是添加嵌入的容器，里面可以继续添加任意的 UI 元素；第三个菜单是添加默认在工具栏中靠右显示的子元素。

3）选择“Insert ToolBarItem”，在弹出的对话框中即可选择相应的元素，如图 4-86 所示。

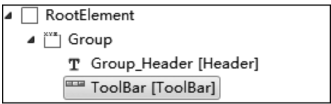


图 4-84　添加后的效果

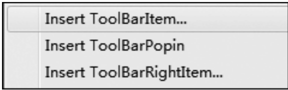


图 4-85　添加子元素的菜单选择

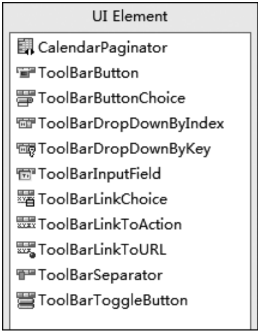


图 4-86　选择子元素的类型

4）选择添加一个工具栏按钮，单击“OK”按钮，按照同样的操作再次添加一个按钮，如图 4-87 所示。

5）保存所有修改，并重新编译、运行之后，就能看到实际的运行效果，如图 4-88 所示。

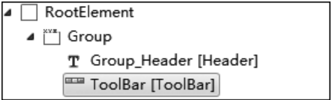


图 4-87　添加后的效果



图 4-88　运行效果

4.2.7 超链接

Web Dynpro 中的超链接包含两种类型："LinkToAction" 和 "LinkToURL"，两种链接都是以网页链接的形式（默认为蓝色带下画线字体）在视图中展示。不同的是动作链接在用户单击后会触发相应的动作，URL 链接则是直接打开属性中配置的网页超链接（更像是 HTML 中的一个链接）。

不管是哪种链接形式，都包含了下列共同属性。

1）ActivateAccessKey：设置是否允许快捷键（使用"Alt"键与首字母触发默认事件）。

2）ContextMenuBehaviour：设置上下文菜单行为，默认有"继承""允许"和"禁止"3 种类型。

3）ContextMenuId：设置上下文菜单的唯一标识。

4）controlMenuDisabling：控制菜单是否可用。

5）Design：控制链接文本的显示风格，默认有标准显示和增强显示两种类型，其中增强类型即为文本加粗显示。

6）Enabled：控制元素是否可用。

7）Hotkey：设置指定键盘组合作为快捷键使用，必须从下拉菜单中选择，不可自定义设置。

8）imageFirst：控制显示的链接图标是否位于文本前面，布尔型属性。

9）imageHeight：控制链接图标的高度。

10）imageSource：控制链接图标的来源（文件目录位置）。

11）imageWidth：控制链接图标的宽度。

12）text：控制链接的显示文本。

13）textDirection：控制链接文本的排列顺序，默认有 3 个选项：继承自父元素、从左往右和从右往左。

14）tooltip：控制提示消息的显示文本（对应老版本中的"imageAlt"属性）。

15）type：控制链接的显示风格，默认有以下 4 种风格：

- 功能型：显示默认的链接风格，默认为蓝色带下画线文本。
- 导航型：显示为已访问过的链接颜色，文本带下画线。
- 报告型：显示默认的链接风格，但文本不带下画线。
- 结果型：显示为已访问过的链接颜色，但文本不带下画线。

16）visible：控制元素是否可见。

17）wrapping：控制文本是否可折叠（页面宽度不足时将文本自动换行）。

而两种链接又分别有一些特有的属性。其中动作链接包含以下特有属性。

- "onAction" 事件：用于用户单击链接时触发的服务器端事件处理器。

URL 链接包含以下特有属性。

- reference：控制单击打开的 URL 地址。
- target：控制弹出页面的具体位置，如可以使弹出页面在指定的窗口中打开。

开发实例：

1）在任意视图中创建一个动作链接，并调整其属性，将"Design"修改为"emphsized"，将"imageFirst"修改为"false"，"imageSource"选择 SAP 标准图标中的"s_b_

urls. gif”，将“text”修改为“动作链接示例”，如图 4-89 所示。

2）编译、运行后的效果如图 4-90 所示。

LinkToAction（General / LayoutData / Events）

design	emphasized
enabled	true
hotkey	none*
imageFirst	false
imageHeight	
imageSource	~sapicons/s_b_urls.gif
imageWidth	
text	动作链接示例
textDirection	inherit*
tooltip	
type	function*
visible	visible*
wrapping	false

图 4-89　修改属性



图 4-90　运行效果

3）在任意视图中创建一个 URL 链接，并调整其属性，“imageSource”选择 SAP 标准图标中的“s_b_anwe. gif”，将“imageWidth”修改为“30px”，将“reference”修改为需要访问的网页地址，将“text”修改为“URL 链接”，如图 4-91 所示。

4）保存所有修改，并重新编译、运行之后，就能看到实际的效果，如图 4-92 所示。

LinkToURL（General / LayoutData）

imageFirst	true
imageHeight	
imageSource	~sapicons/s_b_anwe.gif
imageWidth	30px
reference	http://www.sapeyes.com
target	
text	URL链接
textDirection	ltr
tooltip	
type	navigation*
visible	visible*

图 4-91　修改属性

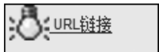


图 4-92　运行效果

4. 2. 8　选项卡集

Web Dynpro 中的选项卡集元素（页签）实际上是一系列选项卡的容器或者集合，它的子元素只能是选项卡（Tab）。它包含了下列属性。

1）accessibilityDescription：设置消息提示和描述等文本是否在用户选中整个选项卡集时显示。

2）ActivateAccessKey：控制是否允许快捷键（使用“Alt”键与首字母触发默认事件）。

3）ContextMenuBehaviour：控制上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。

4）ContextMenuId：上下文菜单唯一标识。

5）enabled：控制元素是否可用。

6）handleHotkeys：设置是否处理快捷键触发事件。

7）height：控制元素的显示高度。

8）selectedTab：控制默认选中的选项卡。

9）selectionChangeBehaviour：设置选项卡切换时是否自动切换选项卡集的首选项（lead selection），有自动或手动两个选项。

10）tabAlignment：设置子选项卡大小的调整方式，默认包含以下两种调整方式：

- exact：按照子选项卡的确切大小进行显示。
- fast：允许客户端更有效地显示而对高度和宽度进行优化。

11）tooltip：控制提示消息的显示文本。

12）visible：控制元素是否可见。

13）width：控制元素的显示宽度。

选项卡集默认有以下两个动作可以定义并触发相应事件。

- onClose：当单击选项卡的关闭按钮时触发相应事件。
- onSelect：当用户进行选项卡切换时触发相应的事件。

作为选项卡集唯一可用的子元素的选项卡，它包含了下列属性。

- closeable：控制选项卡是否可以被用户关闭。
- enabled：控制元素是否可用。
- hasContextPadding：控制选项卡页面内部是否有间距，类似 Word 文档中的页边距。
- visibility：控制元素是否可见。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），首先需要上下文的定义。创建一个集合属性为“1‥1”的结点，随后创建员工信息相关的上下文特性，见表 4-2。

表 4-2　员工信息相关的上下文特性

特性 ID 名称	特 性 描 述	特 性 类 型
PrInfo Node	人员信息	结点
id	用户 ID	字符串
name	姓名	字符串
age	年龄	整数类型
dept	部门	字符串
createDate	创建日期	日期类型

2）创建完毕后，保存所有修改，如图 4-93 所示。

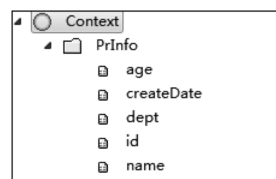


图 4-93　特性清单

3）右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“TabStrip”选项卡集类型，输入 ID 名称，单击“OK”按钮，如图 4-94 所示。

4）如果不添加选项卡，选项卡集只是一个空的容器，在“Outline”中右击选项卡集，选择“Insert Tab”，如图 4-95 所示。

5）操作两次，共新增“Tab”和“Tab1”两个选项卡，如图 4-96 所示。

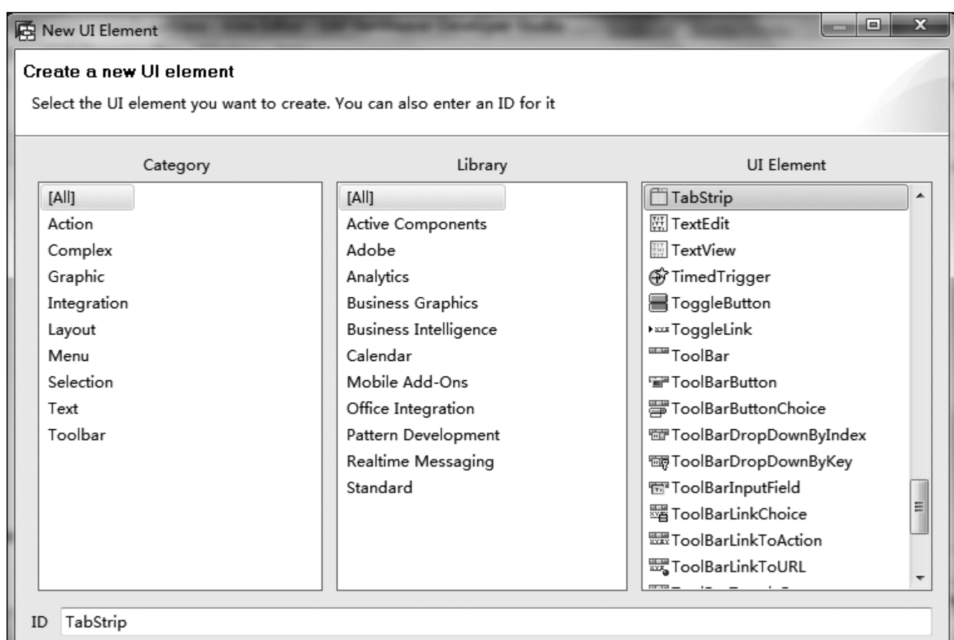


图 4-94　选择元素类型

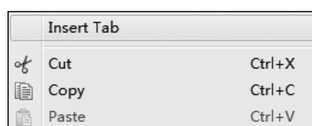


图 4-95　添加选项卡的菜单选择

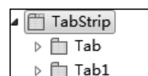


图 4-96　添加后效果

6）将选项卡的“text”属性分别修改为“个人信息概览”和“个人信息明细”，如图 4-97 所示。

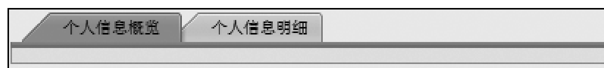


图 4-97　运行效果

7）在第一个选项卡中添加一个标签和 1 个文本框，如图 4-98 所示。

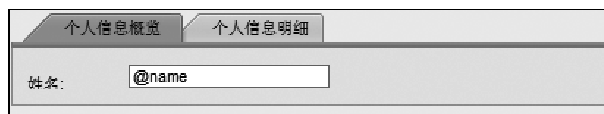


图 4-98　预览效果

8）右击第二个选项卡中的“Tab1_Content”容器并选择“Apply Template”，选择“Form”表单，单击“Next”按钮，如图 4-99 所示。

9）选择表单中需要使用的上下文特性，单击“Next”按钮，如图 4-100 所示。

10）选择页面元素的具体类型，其中“dept”部门使用“DropDownByKey”下拉列表，单击“Finish”按钮，如图 4-101 所示。

11）在选项卡中生成了对应的表单页面，如图 4-102 所示。

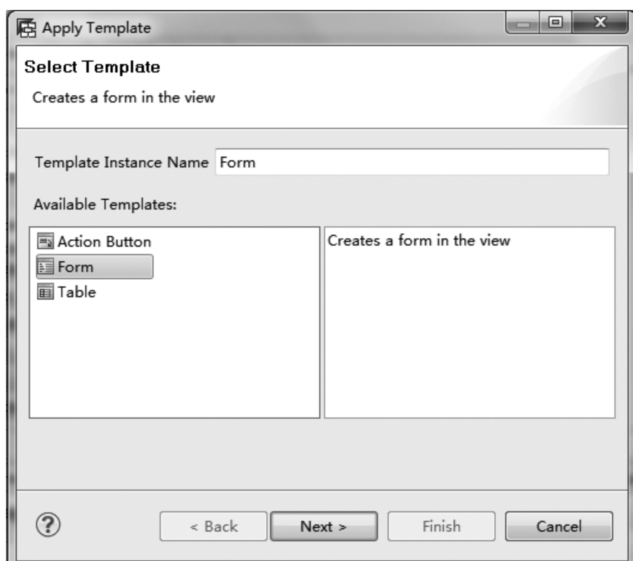


图 4-99　选择模板类型

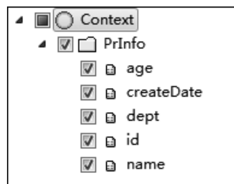


图 4-100　选择对应的上下文



Name	Attribute	Editor	Bound Prop...
id	id (PrInfo.id)	InputField	value
name	name (PrInfo.name)	InputField	value
age	age (PrInfo.age)	InputField	value
dept	dept (PrInfo.dept)	DropDownBy...	selectedKey
createDate	createDate (PrInfo.cre...	InputField	value

图 4-101　调整元素类型和位置

图 4-102　预览效果

12）如果将第一个选项卡的可关闭属性改为“true”，则用户就可以手动关闭对应选项卡，如图 4-103 所示。

13）在配置过程中也可以看到，其实选项卡对应的也只是一个透明容器，所以选项卡中的页面元素调整就可以参考其他的元素介绍自由地进行配置了，如图 4-104 所示。

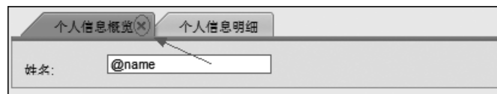


图 4-103　运行效果

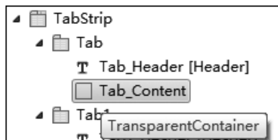


图 4-104　添加后的效果

4.2.9　托盘

托盘是一个类似于组的 UI 元素，它可以将若干的元素包含在特定区域中，并且赋予一个通用的标题，但是它比组元素稍微复杂一些，确切地说，是拥有更多的功能。首先，最直观的区别就是托盘可以被展开或收起，收起时仅显示托盘的标题部分；接着托盘标题右侧可以定义一个菜单按钮进行功能的自定义。托盘中还可以定义工具栏。

托盘包含以下属性。

1）layout：控制元素内部的布局类型。

2）accessibilityDescription：设置消息提示和描述等文本是否在用户选中整个选项卡集时

显示。

3）activateAccessKey：设置是否允许快捷键（使用“Alt”键与首字母触发默认事件）。

4） contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。

5）contextMenuId：设置上下文菜单的唯一标识。

6） defaultButtonId：用于控制默认对应按钮的标识。

7） design：控制托盘元素整体的样式，默认有以下 3 种样式可以选择。

- fill：托盘的内容区域会显示默认的背景颜色。
- plain：托盘的内容区域会显示白色的背景颜色并带有边框。
- transparent：托盘的内容区域背景是透明的，并且内容区域周围没有边框。

8） enabled：控制元素是否可用。

9） expandable：控制托盘是否可以展开或收起。

10） expended：控制托盘的默认状态是否为展开。

11） hanlesHotkeys：控制托盘是否作为一个独立的容器来响应定义的快捷键事件。

12） hasContextPadding：控制托盘的内容区域内部是否与边框有边距。

13） height：控制托盘的高度。

14）scrollingMode：控制滚动条是否在内容区域的容器中可见。

- auto：容器中的滚动条会自动被激活
- both：横向和纵向的滚动条默认已激活
- none：不允许出现滚动条

15）tooltip：控制提示消息的显示文本。

16） visible 控制元素是否可见。

17）width：控制元素的显示宽度。

18）onToggle 事件：托盘的默认事件，会在用户展开或收起托盘时触发。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“Tray”托盘类型，输入 ID 名称，单击“OK”按钮，如图 4-105 所示。

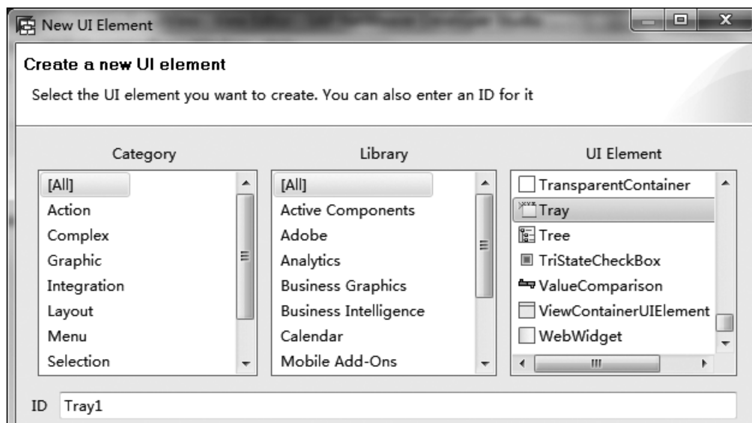


图 4-105　选择元素类型

2）右击“Outline”中的托盘，可以看到托盘中可添加的一些特殊对象，如图 4-106 所示。

3）依次添加工具栏、弹出菜单和透明容器后，就能实现一个相对复杂的需求，如图 4-107 所示。

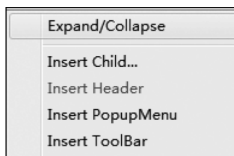


图 4-106　托盘的子元素

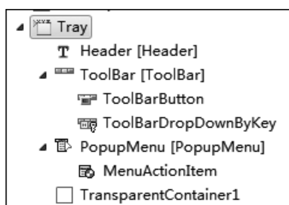


图 4-107　添加后的效果

4）编译、运行后，就能看到实际的视图效果，如图 4-108 所示。

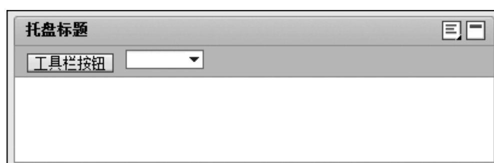


图 4-108　运行效果

4.2.10　消息区域

消息区域是 Web Dynpro 用于定义消息显示的具体位置的元素，包括报错消息、警告消息和成功消息等消息管理器发布的所有消息，同时在同一个视图实例中也仅会存在一个消息区域的实例。如果程序中未定义消息区域，则消息提醒的位置无法统一，当页面元素过多或表单过长时，用户很可能会没注意操作时返回的消息，从而增加用户操作难度以及运维人员的工作量。

消息区域元素包含下列属性。

- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- design：控制链接文本的显示风格，默认有标准显示和增强显示两种类型，其中增强类型即为文本加粗显示。
- enabled：控制元素是否可用。
- maxVisibleMessages：控制消息区域元素能够显示的最大消息数，默认值为“0”表示没有限制。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。

与其他 Web Dynpro 元素不同的是，消息区域的“enabled”属性虽然可以绑定上下文特性，但只能绑定只读状态的上下文，且值为“true”；而消息区域的“visible”属性虽然也可以绑定上下文特性，但也只能绑定只读属性的特性，而且值不能设置为“none”或“blank”。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“MessageArea”，输入ID名称，单击“OK”按钮，如图4-109所示。

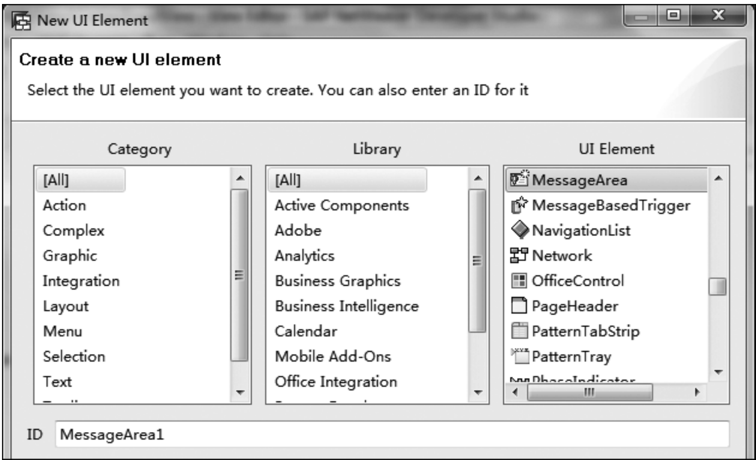


图4-109　选择元素类型

2）在视图中添加一个按钮，用于打印消息。将按钮的“text”文本属性修改为“打印消息”（见图4-110），在按钮的事件中新建一个动作，并添加代码：

```
wdComponentAPI.getMessageManager().reportSuccess("Report Success!");
```

3）保存所有修改，并重新编译、运行，运行效果如图4-111所示。

4）单击“打印消息”按钮，消息显示在顶端，如图4-112所示。

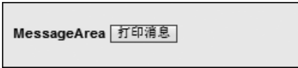


图4-110　预览效果



图4-111　运行效果

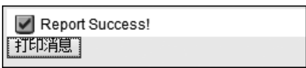


图4-112　单击“打印消息”按钮效果

5）接着回到视图中，并改变消息区域的位置，调整到按钮后面，如图4-113所示。

6）在开发工具中的视图预览效果，如图4-114所示。

7）再次编译、运行之后，单击“打印消息”按钮，消息显示在按钮下方（在不同的开发程序中，会根据实际容器的布局显示在后面或下方），如图4-115所示。



图4-113　调整后的效果



图4-114　预览效果

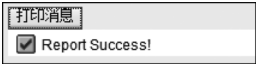


图4-115　运行效果

4.2.11　表单

在介绍了一些常用的简单元素之后，当开发人员将上述元素组合在一起展现在前台时，就形成了表单。表单是需要有样式的，这个可以通过不同的布局来实现。与单个元素创建并构建表单不同，Web Dynpro同样提供了一个向导功能用于表单的快速创建。在熟悉以后，

开发人员也可以根据实际需要进行表单的手动创建。

严格来说，表单并不属于 Web Dynpro 的界面元素，它其实是从业务角度的一个定义，所以在 Web Dynpro 中，一个表单并没有固定属性。它可以根据需要放入不同的容器中，如透明容器、组或托盘等元素，根据这些特定的容器，表单也可以进行一些特殊的设置和调整。

开发实例：

假设需要创建一个表单用于提交员工的基本信息。

1）在创建一个视图后，首先需要上下文的定义。创建一个集合属性为“1‥1”的结点，然后创建与员工信息相关的上下文特性，见表 4-3。

表 4-3　员工信息相关的上下文特性

特性 ID 名称	特 性 描 述	特 性 类 型
PrInfo Node	人员信息	结点
id	用户 ID	字符串
name	姓名	字符串
age	年龄	整数类型
dept	部门	字符串
createDate	创建日期	日期类型

2）创建完毕后，保存所有修改，如图 4-116 所示。

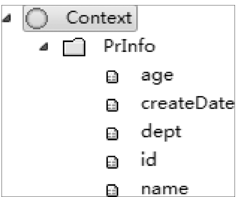


图 4-116　添加特性后的效果

3）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“Group”，输入 ID 名称“V01_Gp”，单击“OK”按钮，如图 4-117 所示。

4）完成创建后，在“Outline”中多了一个组的元素，如图 4-118 所示。

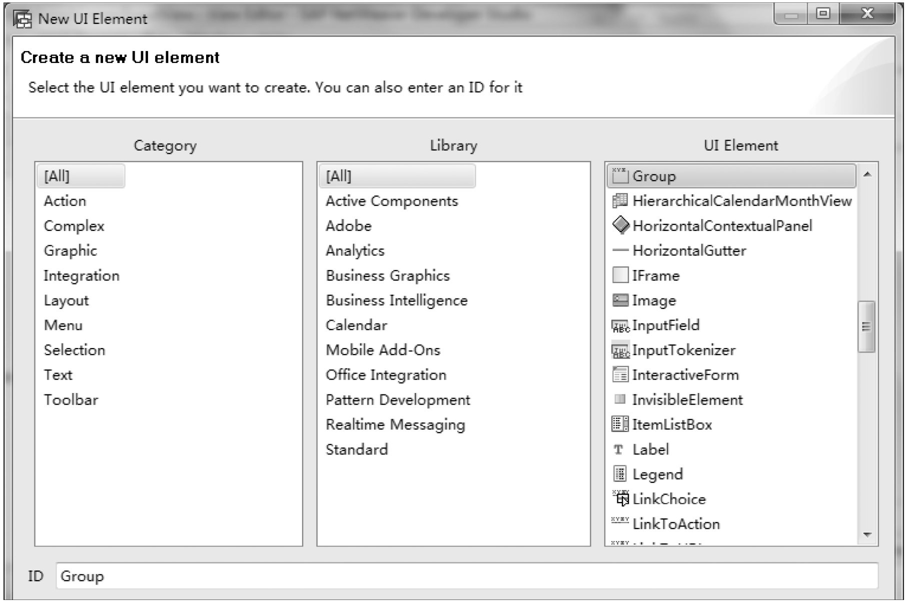


图 4-117　选择元素类型

5）右击“V01_Gp”并选择“Apply Template”，然后选择“Form”表单，单击“Next”按钮，如图 4-119 所示。



图 4-118　添加后的效果

6）随后选择表单中需要使用的上下文特性，单击“Next”按钮，如图 4-120 所示。

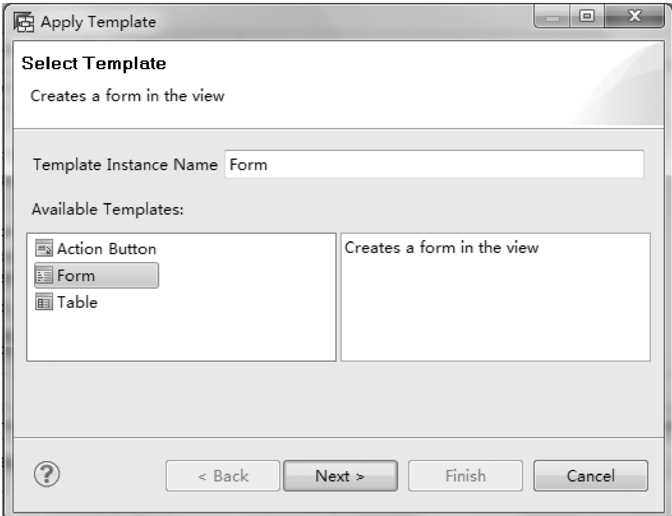


图 4-119　选择模板类型

7）选择页面元素的具体类型，其中“dept”部门使用“DropDownByKey”下拉列表，单击“Finish”按钮，如图 4-121 所示。

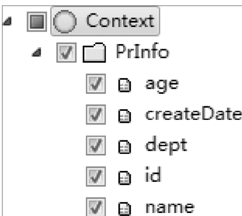


图 4-120　选择对应特性

Name	Attribute	Editor	Bound Prop...
id	id (PrInfo.id)	InputField	value
name	name (PrInfo.name)	InputField	value
age	age (PrInfo.age)	InputField	value
dept	dept (PrInfo.dept)	DropDownBy...	selectedKey
createDate	createDate (PrInfo.cre...	InputField	value

图 4-121　调整元素类型和位置

8）通过向导添加表单后，会在“Outline”的清单中看到一个自动生成的透明容器，里面包含了刚才选择创建的标签和输入框等各种元素，如图 4-122 所示。

9）修改组的属性，设置它的“width”宽度为 100%，定义“Caption”标题内容“人员信息”，如图 4-123 所示。

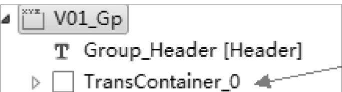


图 4-122　添加后的效果

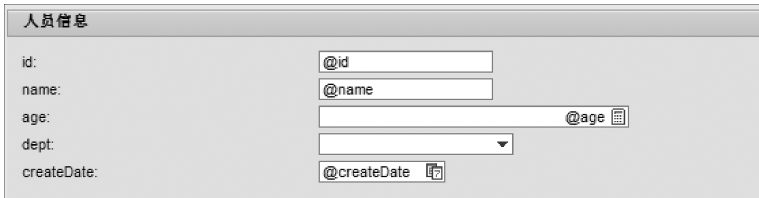


图 4-123　预览效果

10）进一步优化页面布局，调整透明容器的宽度，也可根据实际需求添加“提交”按钮，如图 4-124 所示。

图 4-124　预览效果

11）保存所有修改，并重新编译、运行之后，就能看到实际的效果，如图 4-125 所示。

图 4-125　运行效果

4.3　复杂元素

4.3.1　复选框组

复选框组（CheckBoxGroup）是多个复选框的一个集合，并以一个表格的方式将若干复选框在视图中进行显示，用户可以从这个集合中勾选对应的条目。与单选按钮组不同的是，复选框组中可以进行条目的多选。它包含了下列属性。

- accessibilityDescription：设置复选框组中的文本是否在用户选中整个元素时显示。
- activateAccessKey：设置是否允许快捷键（使用“Alt”键与首字母触发默认事件）。
- colCount：控制每一行显示的复选框的个数。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- enabled：控制元素是否可用。
- explanation：设置元素的解释文本，设置后元素文本下方会出现一条绿色的下画线。当鼠标悬停在文本上时，会弹出解释文本的说明。
- readOnly：控制元素是否只读。
- state：控制元素的显示状态，默认有“normal”和“required”两种状态。在选择“required”后，元素会出现一个红色星号进行提醒，但逻辑控制仍需要编写相关代码。
- textDirection：控制链接文本的排列顺序，默认有 3 个选项：继承自父元素、从左往右和从右往左。
- texts：控制复选框组的显示文本，不能硬编码设置，只能绑定选择属性为“0‥n”或“1‥n”的上下文结点中的特性。

- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- width：控制元素的显示宽度。
- onToggle 事件：复选框组的默认事件，会在用户单击复选框时触发。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“CheckBoxGroup”复选框组类型，输入 ID 名称，单击“OK”按钮，如图 4–126 所示。

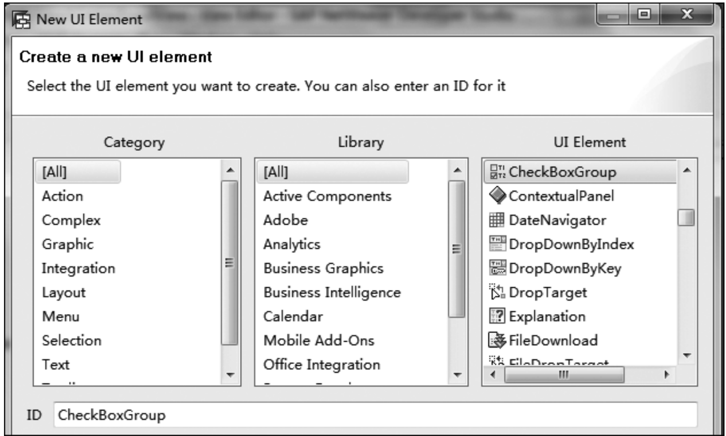


图 4–126　选择元素类型

2）创建对应的上下文结点以及特性，如图 4–127 所示。

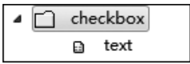


图 4–127　特性清单

3）注意结点的属性中的选择属性必须是“0‥n”或“1‥n”，即允许与之绑定的视图元素选择多个条目，如图 4–128 所示。

Collection Cardinality	0..n
Initialize Lead Selection	true
Name	checkbox
Selection Cardinality	0..n
Singleton	true

图 4–128　结点的属性修改

4）进入 Java 编辑器，编写初始化代码，录入一些模拟选项：

```
ICheckboxElement createAndAddCheckboxElement = wdContext. nodeCheckbox()
  . createAndAddCheckboxElement();//创建选项
wdContext. nodeCheckbox(). setSelected(0, true);//选项是否被选中
  createAndAddCheckboxElement. setText("费用报销");//备注
createAndAddCheckboxElement = wdContext. nodeCheckbox()
  . createAndAddCheckboxElement();
createAndAddCheckboxElement. setText("请假申请");
createAndAddCheckboxElement = wdContext. nodeCheckbox()
  . createAndAddCheckboxElement();
createAndAddCheckboxElement. setText("费用立项");
```

5）选择创建完成的复选框组，将“text”属性与上下文的特性进行绑定，如图 4–129 所示。

6）绑定上下文后，创建一个默认动作“MakeSelection”，对用户的动作进行响应，如图 4-130所示。



图 4-129　修改文本属性



图 4-130　修改事件属性

7）在创建动作时添加两个参数，参数类型分别为布尔型和整数型，用于区分是否勾选以及勾选条目的索引，如图 4-131 所示。

Details　Parameters

Name	Type
checked	boolean
index	integer

图 4-131　添加方法中的参数

8）保存所有修改，右击“Outline”清单中的复选框组，选择“Parameter Mapping”（如果在创建动作时不添加对应参数，则无法进行映射），将复选框组所选的关键字与创建的动作中的参数进行映射，如图 4-132 所示。

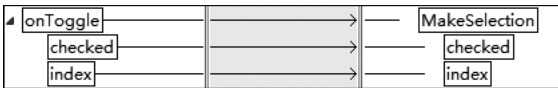


图 4-132　进行参数映射

9）保存所有修改，随后进入 Java 编辑器，在动作对应的方法中添加以下代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess( checked) ;   //打印传递的选中参数
wdComponentAPI. getMessageManager( ). reportSuccess( index) ;     //打印当前元素的索引
```

10）保存所有修改，并重新编译、运行之后即可进行勾选动作的测试，如图 4-133 所示。

11）在勾选了复选框之后，消息区域中也会打印出对应的提示消息。此例子中的消息分别表示已勾选和勾选的索引。索引序号是从“0”开始计算，而图 4-133 最后勾选的是第二个复选框，所以在图 4-134 中看到的索引是“1”，如图 4-134 所示。

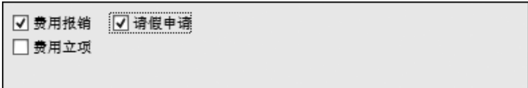


图 4-133　运行效果



图 4-134　选择后的效果

4.3.2　单选按钮组

单选按钮组（RadioButtonGroupByKey）是将若干单选按钮放入一个表中合成的一个元素。与复选框组不同的是，单选按钮组只允许选中一个条目。它包含下列属性。

- accessibilityDescription：设置单选按钮组中的文本是否在用户选中整个元素时显示。
- activateAccessKey：设置是否允许快捷键（使用“Alt”键与首字母触发默认事件）。
- colCount：控制每一行显示的单选按钮的个数。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。

- enabled：控制元素是否可用。
- explanation：设置元素的解释文本，设置后元素文本下方会出现一条绿色的下画线。当鼠标悬停在文本上时，会弹出解释文本的说明。
- readOnly：控制元素是否只读。
- selectedKey：指定上下文特性用于存放当前选中的条目的关键字，必须绑定上下文。
- state：控制元素的显示状态，默认有“normal”和“required”两种状态。在选择“required”后，元素会出现一个红色星号进行提醒，但逻辑控制仍需要编写相关代码。
- textDirection：控制链接文本的排列顺序，默认有3个选项：继承自父元素、从左往右和从右往左。
- texts：控制单选按钮组的显示文本，不能硬编码设置，只能绑定选择属性为“0‥n”或“1‥n”的上下文结点中的特性。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- width：控制元素的显示宽度。
- onSelect 事件：单选按钮组的默认事件，会在用户单击按钮时触发。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择单选按钮组类型，输入 ID 名称，单击“OK”按钮，如图 4-135 所示。

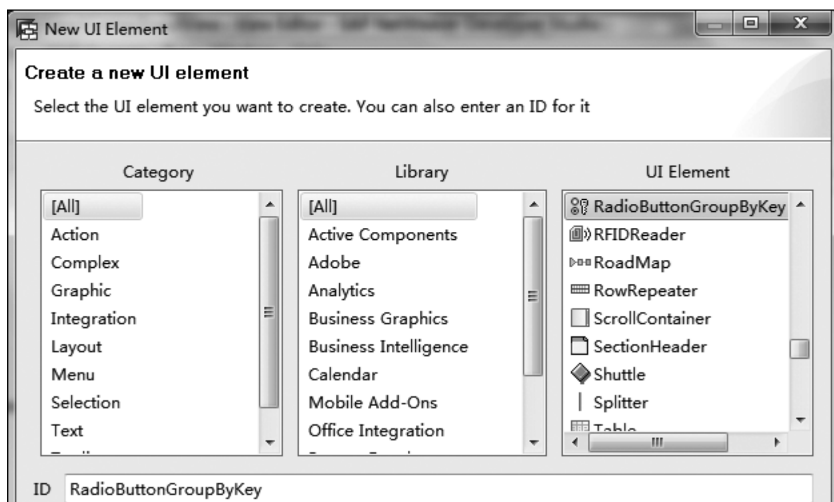


图 4-135　选择元素类型

2）打开视图的上下文页签，创建一个特性，如图 4-136 所示。



图 4-136　特性清单

3）该特性使用的是 mydynp04 中创建的简单类型，即包含了 3 个部门的数据和一条空数据，如图 4-137 所示。

4）随后打开单选按钮组的属性，将“selectedKey”属性与上下文的特性进行绑定，如图 4-138 所示。

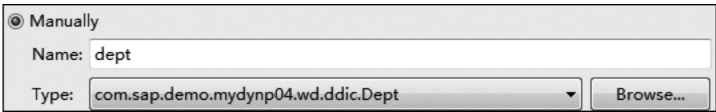


图 4-137　创建特性

5）绑定上下文后，创建一个默认动作“select”对用户的动作进行响应，如图 4-139 所示。



图 4-138　修改属性　　　　图 4-139　修改事件属性

6）在创建动作时添加一个字符串类型的参数，用于记录在单选按钮组中选中的关键字，如图 4-140 所示。

Details | Parameters

Name	Type
key	string

图 4-140　添加参数

7）保存所有修改，右击“Outline”清单中的单选按钮组，选择“Parameter Mapping”（如果在创建动作时不添加对应参数，则无法进行映射），将单选按钮组所选的关键字与创建的动作中的参数进行映射，如图 4-141 所示。

12）保存所有修改，随后进入 Java 编辑器在动作对应的方法中添加以下代码：

```
wdComponentAPI. getMessageManager( ). reportSuccess( "key:" + key);          //打印选中条目
的关键字
wdComponentAPI. getMessageManager( ). reportSuccess(
"dept:" + wdContext. currentContextElement( ). getDept( ));                   //打印部门属性
```

8）保存所有修改，并重新编译，运行之后即可进行勾选动作的测试，如图 4-142 所示。

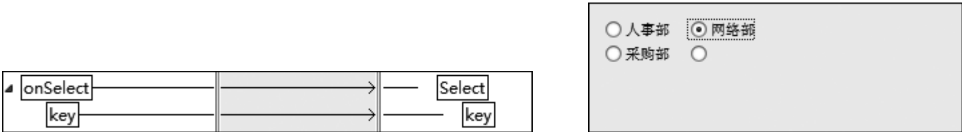


图 4-141　进行参数映射　　　　图 4-142　运行效果

9）在选中任一个单选按钮之后，消息区域中也会打印出对应的提示消息。此例子中的消息分别表示选中的关键字以及上下文特性的对应值。由于在简单类型创建中，“网络部”对应的关键字是“02”，因此在图 4-143中看到的消息均为“02”，如图 4-143 所示。



图 4-143　选择后效果

4.3.3　表格

WD4J 技术对表格做了更充分的封装，开发人员不用关心表格控件本身，只用专注于绑定数据源的操作。表格设计时的生成建议采用模板导航自动生成，并且在通过上下文结构生成表格列时，也可以根据特性的类型选择不同的界面元素进行个性化显示。

表格包含以下属性。

1）dataSource：表格对应的上下文结点数据源，必须绑定上下文。

2）fixedTableLayout：控制表格布局是否固定，当设置为“false”时，表格的列可以自由拖动并变更位置。

3）multiColSorting：控制是否支持复合列排序。

4）readOnly：控制表格是否只读。

5）rowSelectable：控制表格行项目是否可选。

6）scrollColCount：控制滚动条显示的列数目。

7）selectionMode：控制表格的选择模式，默认有“multi”“multiNoLead”“none”和“single”4个类型。

- multi：表格可以多选，并且默认首行选中。
- multiNoLead：表格可以多选，默认不选中首行。
- none：表格既不可以单选也不可以多选。
- single：表格只可以对行项目进行单选。

注：设置为“multi”与“multiNodeLead”还有一个前提，就是绑定的上下文结点支持多条目的选择，即结点的可选属性为“0‥n”或“1‥n”。

8）visible：控制表格是否可见。

9）visibleRowCount：表格可见行项目数。

10）width：控制表格宽度。

11）onColSelect 事件：选中任意列时触发。

12）onDrop 事件：拖动列时触发。

13）onFiltet 事件：用于表格过滤。

14）onLeadSelect 事件：当首选项发生变化时触发，对应的表格行项目只能单选。

15）onSelect 事件：选中行项目时触发，对应的表格行项目可以单选或多选。

16）onSort 事件：用于表格排序。

下面通过一个表格的开发实例，来了解表格的功能和实际作用。

1. 创建表格

首先在视图中通过应用模板的方式创建一个表格。

1）与表单的创建方式相同，表格也可以通过应用模板的方式进行创建。首先在视图上下文中创建一个集合属性为“0‥n”的结点“PerLst”，如图4-144所示。

2）右击视图中的“RootElement”根结点元素，选择“Apply Template”，在弹出的对话框中选择“Table”表格类型，单击“Next”按钮，然后选择并绑定上下文数据源，如图4-145所示。

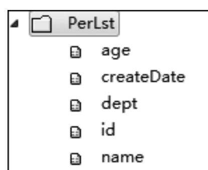


图4-144　特性清单

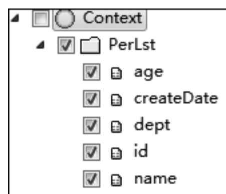
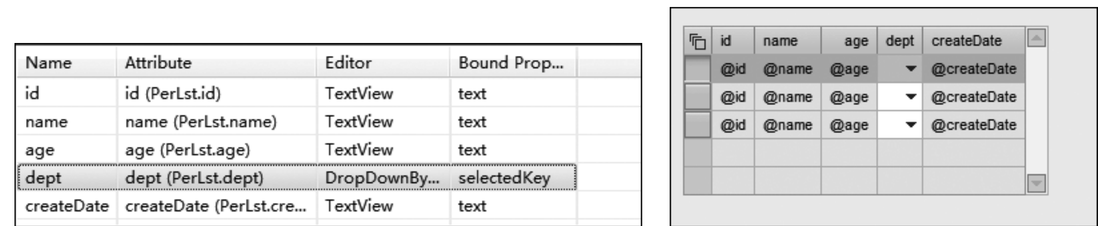


图4-145　选择对应上下文

3）接着选择不同字段在表格列中的界面元素类型，将其中的“dept”字段用下拉列表的方式进行展现，单击“Finish”按钮，如图 4-146 所示。

4）完成表格创建后，视图中就能看到自动生成的表格样式，如图 4-147 所示。



Name	Attribute	Editor	Bound Prop...
id	id (PerLst.id)	TextView	text
name	name (PerLst.name)	TextView	text
age	age (PerLst.age)	TextView	text
dept	dept (PerLst.dept)	DropDownBy...	selectedKey
createDate	createDate (PerLst.cre...	TextView	text

图 4-146　调整元素类型和位置

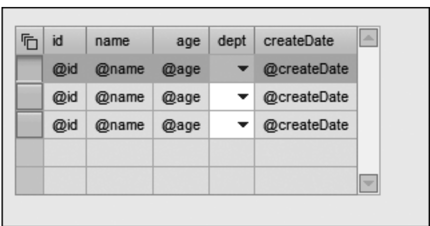


图 4-147　预览效果

5）对表格做进一步的布局优化，插入标题，并将表格的列名称改为实际表格名称，同时设置表格宽度与列宽度，如图 4-148 所示。



图 4-148　调整属性后的效果

6）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-149 所示。

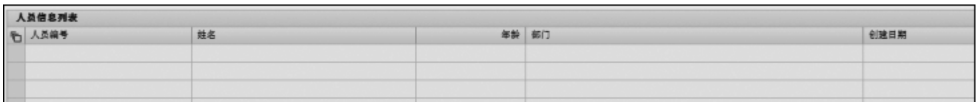


图 4-149　运行效果

7）先初始化几条数据。在视图的“wdDoInit”方法中添加以下测试代码：

```
IPerLstNode nodePerLst = wdContext. nodePerLst( ) ;
java. sql. Date date = new java. sql. Date( System. currentTimeMillis( ) ) ;
IPerLstElement createAndAddPerLstElement = nodePerLst. createAndAddPerLstElement( ) ;
createAndAddPerLstElement. setId( "001" ) ;
createAndAddPerLstElement. setName( "李 1" ) ;
createAndAddPerLstElement. setAge(30) ;
createAndAddPerLstElement. setDept( "01" ) ;
createAndAddPerLstElement. setCreateDate( date) ;
createAndAddPerLstElement = nodePerLst. createAndAddPerLstElement( ) ;
createAndAddPerLstElement. setId( "002" ) ;
createAndAddPerLstElement. setName( "李 2" ) ;
createAndAddPerLstElement. setAge(31) ;
createAndAddPerLstElement. setDept( "02" ) ;
createAndAddPerLstElement. setCreateDate( date) ;
```

8）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-150 所示。注意，部门的类型仍然是之前建立的“Dept”简单类型，所以在初始化赋值之后，下

拉列表框显示了数据字典中简单类型对应的描述。

人员信息列表

人员编号	姓名	年龄	部门	创建日期
001	李1	30	人事部	2013-8-27
002	李2	31	网络部	2013-8-27

图 4-150　运行效果

2. 设置属性和事件

（1）表格属性

在完成表格的创建之后，继续对视图元素进行属性的个性化。

1）修改部门列属性，将编辑框的只读设置为“true”，如图 4-151所示。

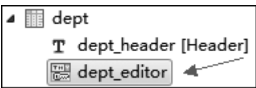


图 4-151　添加后的效果

2）保存所有修改，并重新进行编译、运行。由于表格绑定的“PerLst”结点的可选集合为“0··1”，因此默认情况下表格仅持单选，不支持多选，如图 4-152 所示。

人员信息列表

人员编号	姓名	年龄	部门	创建日期
001	李1	30	人事部	2013-8-27
002	李2	31	网络部	2013-8-27

图 4-152　单行选择运行效果

3）如果将表格的“Initialize Lead Selection”设置为“false”，则运行后视图中的表格不会有默认选中行，如图 4-153 所示。

人员信息列表

人员编号	姓名	年龄	部门	创建日期
001	李1	30	人事部	2013-8-27
002	李2	31	网络部	2013-8-27

图 4-153　不可选运行效果

4）当“PerLst”结点的可选属性为“0··n”或“1··n”时，表格支持多选，如图 4-154 所示。

人员信息列表

人员编号	姓名	年龄	部门	创建日期
001	李1	30	人事部	2013-8-27
002	李2	31	网络部	2013-8-27

图 4-154　多选运行效果

5）接着将表格的“selectionMode”属性修改为“single”，随后编译并运行新程序，如图 4-155 所示。



图 4-155　修改属性

6）此时虽然上下文结点“PerLst”仍然支持多选，但由于表格属性的设置，表格还是只支持单选，如图 4-156 所示。

人员信息列表

人员编号	姓名	年龄	部门	创建日期
001	李1	30	人事部	2013-8-27
002	李2	31	网络部	2013-8-27

图 4-156　运行效果

7）设置表格为只读，将“rowSelectable”修改为“false”，如图 4-157 所示。

readOnly	true
rowSelectable	false

图 4-157　修改属性

8）重新编译并运行之后，表格行项目将无法被选中，如图 4-158 所示。

人员信息列表

人员编号	姓名	年龄	部门	创建日期
001	李1	30	人事部	2013-8-27
002	李2	31	网络部	2013-8-27

图 4-158　运行效果

（2）表格事件

下面通过实例来了解表格中的各个事件定义和开发。

1）创建表格默认事件“ColSelect”，如图 4-159 所示。

Events	
onColSelect	ColSelect

图 4-159　修改事件属性

2）在创建该自定义动作时，添加两个参数“col”和“ele”，用于处理选中行的序号以及表格的行项目元素，如图 4-160 所示。

Details　Parameters

Name	Type
col	string
ele	com.sap.tc.webdynpro.progmodel.api.IWDNodeElement

图 4-160　添加参数

3）两个参数分别需要与表格元素进行参数映射，这样在触发相应事件时才会自动接收表格元素传递的标准参数。在“Outline”中右击“Table”，并选择“Parameter Mapping”，如图 4-161 所示。

4）在弹出的对话框中将表格元素的参数与动作参数进行映射，如图 4-162 所示。

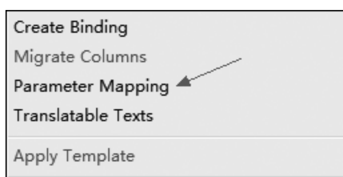


图 4-161　进行参数映射的菜单选择

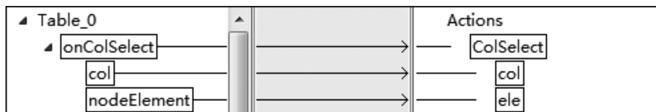


图 4-162　进行参数映射

5）设置表格列的“colSelectionState”为“notSelected”，如图 4-163 所示。

6）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-164 所示。

colSelectionState	notSelected

图 4-163　修改属性

人员信息列表

人员编号	姓名	年龄	部门
001	李1	30	人事部
002	李2	31	网络部

图 4-164　运行效果

7）选择任意的人员编号行后，打印出对应的消息，证明已获取到表格列的标识，如图 4-165所示。

8）接着创建“onLeadSelect”的动作并用同样的方法添加参数，如图 4-166 所示。

所在列：id行项目：null

图 4-165　选择后的效果

onLeadSelect	LeadSelect

图 4-166　修改属性

9）在创建该自定义动作时，添加两个参数“col”和“ele”，用于处理选中行的序号以及表格的行项目元素，如图 4-167 所示。

Details　Parameters

Name	Type
col	string
ele	com.sap.tc.webdynpro.progmodel.api.IWDNodeElement

图 4-167　添加参数

10）右击表格，使用同样的方式进行参数映射，如图 4-168 所示。

11）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-169 所示。

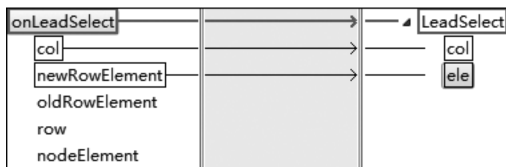


图 4-168　进行参数映射

人员信息列表

人员编号	姓名	年龄	部门
001	李1	30	人事部
002	李2	31	网络部

图 4-169　运行效果

12）当选中任意一行时，打印出对应的行项目信息，如图 4-170 所示。

13）接着创建“onSelect”动作，如图 4-171 所示。

行项目：NodeElement(Dynp04_01Comp/Dynp04_01Comp_05View.PerLst.1)

图 4-170　选择后的效果

onLeadSelect	
onSelect	Select

图 4-171　修改事件属性

注："onSelect"与"onLeadSelect"动作是不能同时定义的，因为前者包含了后者的功能，所以去掉了之前设置的 onLeadSelection。

14）添加自定义动作中的参数，如图 4-172 所示。

Name	Type
newLead	string
oldLead	string
ele	com.sap.tc.webdynpro.progmodel.api.IWDNodeElement

图 4-172　添加参数

15）右击表格，使用同样的方式进行参数映射，如图 4-173 所示。

16）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-174 所示。

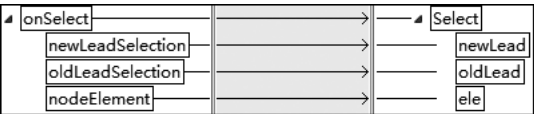


图 4-173　进行参数映射

人员信息列表

人员编号	姓名	年龄	部门
001	李1	30	人事部
002	李2	31	网络部

图 4-174　运行效果

17）选中任意行后，打印出对应的消息，如图 4-175 所示。

旧选项：CMHF.Dynp04_01Comp_05View.PerLst.0新选项：CMHF.Dynp04_01Comp_05View.PerLst.1

图 4-175　选择后的效果

3. 过滤与排序功能

表格提供了标准的过滤与排序类，对于自定义表格可以通过简单封装实现过滤与排序。

表格过滤类参考代码如下：

```
public final class TableFilter{
    public TableFilter(IWDTable table,IWDAction filterAction,
            IWDNode sourceNode,Hashtable hashicons){
        init(table,filterAction,sourceNode,hashicons);
    }
    private void init(IWDTable table,IWDAction filterAction,
            IWDNode sourceNode,Hashtable hashicons){
        this.table = table;
        if (sourceNode == null)
            throw new IllegalArgumentException("SourceNodemust be given");
        if (filterAction == null)
            throw new IllegalArgumentException("Filter action must be given");
        if (table == null)
            throw new IllegalArgumentException("Table must be given");
        if (table.bindingOfDataSource() == null)
            throw new IllegalArgumentException("Data source of table with id'  "
                    + table.getId() + " must be bound");
        this.sourceNode = sourceNode;
        if (hashicons != null){
            this.hashIcon = new Hashtable();
            this.hashIcon = hashicons;
        }
        String dataSourcePrefix = table.bindingOfDataSource() + ".";
```

```
            //works on columns
            setFiltersForColumns( dataSourcePrefix,table. iterateColumns( ) ,
                    sourceNode) ;
            setFiltersForColumns( dataSourcePrefix,table. iterateGroupedColumns( ) ,
                    sourceNode) ;
            table. setOnFilter( filterAction) ;
            filterAction. setEnabled( true) ;
            this. sourceNode = sourceNode;
        }
        private void setFiltersForColumns( String dataSourcePrefix,
                Iterator columnIterator,IWDNode sourceNode) {
            attributeSource = new Hashtable( ) ;
            for ( Iterator iter = sourceNode. getNodeInfo( ) . iterateAttributes( ) ; iter
                    . hasNext( ) ; ) {
                IWDAttributeInfo attribInfo = ( IWDAttributeInfo)  iter. next( ) ;
                String attribName = attribInfo. getName( ) ;
                attributeSource. put( attribName,attribInfo. getSimpleType( )
                        . getBuiltInType( ) ) ;
            }
            int index =0;
            for ( Iterator it = columnIterator; it. hasNext( ) ; ++index) {
                IWDAbstractTableColumn abstractColumn = ( IWDAbstractTableColumn)  it
                        . next( ) ;
                if ( abstractColumn instanceof IWDTableColumn) {
                    IWDTableColumn column = ( IWDTableColumn)  abstractColumn;
                    String columnId = column. getId( ) ;
                    Comparator comparator = null;
                    NodeElementByAttributeComparator elementComparator = null;
                    String bindingOfPrimaryProperty =
bindingOfPrimaryProperty( column
                            . getTableCellEditor( ) ) ;
                    String attributeName = null;
                    if ( bindingOfPrimaryProperty == null
                            || ! bindingOfPrimaryProperty
                                    . startsWith( dataSourcePrefix) ) {
                        if ( hashIcon == null) {
                            continue;
                        } else{
                            if ( ! hashIcon. containsKey( columnId) ) {
                                continue;
                            } else{
                                attributeName = hashIcon. get( columnId) . toString( ) ;
                            }
                        }
                    } else{
                        attributeName = bindingOfPrimaryProperty
                                . substring( dataSourcePrefix. length( ) ) ;
                        if ( hashIcon. containsKey( columnId) )
                            attributeName = hashIcon. get( columnId) . toString( ) ;
                    }
                    String attributeInfo = null;
                    if ( attributeSource. containsKey( attributeName) )
                    attributeInfo = ( String)
attributeSource. get( attributeName) ;
                    elementComparator = new NodeElementByAttributeComparator(
```

```
                    attributeInfo,column,comparator);
                    filterForColumn.put(attributeName,elementComparator);
                } else if (abstractColumn instanceof IWDTableColumnGroup) {
                    IWDTableColumnGroup columnGroup = (IWDTableColumnGroup)
abstractColumn;
                    setFiltersForColumns(dataSourcePrefix,columnGroup
                            .iterateColumns(),sourceNode);
                }
            }
        }
        public void filter(IWDNode allDataNode,IWDNode targetNode) {
            if (allDataNode != null) {
                this.sourceNode = allDataNode;
            }
            HashMap FilterValuesByAttribute = prepareFilterValuesByAttribute();
            ArrayList ar = new ArrayList();
            for (int i = 0; i < sourceNode.size(); i++) {
                IWDNodeElement el = sourceNode.getElementAt(i);
                boolean addelement = true;
                for (Enumeration e = this.attributeSource.keys(); e
                        .hasMoreElements();) {
                    String attributeName = e.nextElement().toString();
                    Object o1 = el.getAttributeValue(attributeName);
                    if (FilterValuesByAttribute.containsKey(attributeName)
                            &&filterForColumn.containsKey(attributeName)) {
                        NodeElementByAttributeComparator elementComparator = (NodeElementB-
yAttributeComparator) filterForColumn
                                .get(attributeName);
                        HashMap valuehash = new HashMap();
                        valuehash = (HashMap) FilterValuesByAttribute
                                .get(attributeName);
                        String attributetype = elementComparator.attributeInfo;
                        Comparator comparator = elementComparator.comparator;
                        if (valuehash.size() == 0) {
                            elementComparator.column.setIsFiltered(false);
                        } else {
                            elementComparator.column.setIsFiltered(true);
                        }
                        addelement = this.evaluateFilteredAttributeValue(valuehash,
                                attributetype,comparator,o1);
                        if (! addelement)
                            break;
                    }
                    if (! addelement)
                        break;
                }
                if (! addelement)
                    continue;
                IWDNodeElement targetElement = targetNode.createElement();
                WDCopyService.copyCorresponding(el,targetElement);
                ar.add(targetElement);
            }
            targetNode.bind(ar);
        }
        public void deleteAllDataNodeElement(IWDNode sourceNode,
```

```
            IWDNode targetNode,String attributeRowId,ArrayList idsToDelete){
        for (int j = sourceNode.size() - 1; j >= 0; --j){
            IWDNodeElement els = sourceNode.getElementAt(j);
            String elsid = els.getAttributeAsText(attributeRowId);
            if (! idsToDelete.contains(elsid))
                continue;
            sourceNode.removeElement(els);
        }
        for (int i = targetNode.size() - 1; i >= 0; --i){
            IWDNodeElement elt = targetNode.getElementAt(i);
            String eltid = elt.getAttributeAsText(attributeRowId);
            if (! idsToDelete.contains(eltid))
                continue;
            targetNode.removeElement(elt);
        }
    }
    /*
     * Update all data node if the table node has been updated if a row has been
     * added in the table node,the all data node can also be updated by using
     * the using the flag addRowOnFly
     */
    public void updateAllDataNodeElement(IWDNode sourceNode,
            IWDNode targetNode,String attributeRowId,boolean addRowOnFly){
        HashMap hashs = new HashMap();
        HashMap hasht = new HashMap();
        HashMap hashup = new HashMap();
        IWDNodeElement elt;
        IWDNodeElement els;
        for (int j = sourceNode.size() - 1; j >= 0; --j){
            els = sourceNode.getElementAt(j);
            String elsid = els.getAttributeAsText(attributeRowId);
            hashs.put(elsid,els);
        }
        for (int i = 0; i < targetNode.size(); i++){
            elt = targetNode.getElementAt(i);
            String rowid = elt.getAttributeAsText(attributeRowId);
            boolean isInSource = hashs.containsKey(rowid);
            if (isInSource && ! elt.isChangedByClient())
                continue;
            if (! isInSource){
                els = sourceNode.createElement();
                WDCopyService.copyCorresponding(elt,els);
                sourceNode.addElement(els);
            } else{
                els = (IWDNodeElement) hashs.get(rowid);
                WDCopyService.copyCorresponding(elt,els);
            }
        }
    }
    /*
     * find if an column has a valid filter and if so parse the filter and put
     * in a hash table
     */
    private HashMap prepareFilterValuesByAttribute(){
        HashMap hashmap = new HashMap();
```

```
        for (Enumeration e = this. attributeSource. keys( ); e. hasMoreElements( ); ) {
            String attributeName = e. nextElement( ). toString( );
            if (filterForColumn. containsKey(attributeName)) {
                NodeElementByAttributeComparator elementComparator = (NodeElementByAt-
tributeComparator) filterForColumn
                        . get(attributeName);
                IWDTableColumn column = elementComparator. column;
                column. setIsFiltered(false);
                String filtervalue = column. getFilterValue( );
                if (filtervalue == null || filtervalue. trim( ). length( ) == 0) {
                    continue;
                }
                String attributetype = elementComparator. attributeInfo;
                HashMap valuehash = new HashMap( );
                valuehash = this. detectFilterSign(filtervalue);
                hashmap. put(attributeName, valuehash);
            }
        }
        return hashmap;
    }
    /* generate a filter value into the right object */
    private Object generateObject(String filtervalue, String attributetype,
            String sign) {
        Object o2 = filtervalue;
        try {
            if (filtervalue == null) {
                return o2;
            }
            if (attributetype. equalsIgnoreCase("decimal")) {
                o2 = new BigDecimal(filtervalue);
            }
            if (attributetype. equalsIgnoreCase("double")) {
                o2 = new Double((filtervalue));
            }
            if (attributetype. equalsIgnoreCase("long")) {
                o2 = new Long(filtervalue);
            }
            if (attributetype. equalsIgnoreCase("float")) {
                o2 = new Float(filtervalue);
            }
            if (attributetype. equalsIgnoreCase("short")) {
                o2 = new Short(filtervalue);
            }
            if (attributetype. equalsIgnoreCase("integer")) {
                o2 = new Integer(filtervalue);
            }
            if (attributetype. equalsIgnoreCase("long")) {
                o2 = new Long(filtervalue);
            }
            if (attributetype. equalsIgnoreCase("date")) {
                try {
                    java. util. Date dtTmp = new java. util. Date(filtervalue);
                    SimpleDateFormat template = new SimpleDateFormat(
                            "yyyy - MM - dd");
```

```
                    o2 = String.valueOf(template.format(dtTmp)); //Date.valueOf(template.
format(dtTmp));
                } catch (Exception ex){
                    o2 = filtervalue;
                }
            }
            if (attributetype.equalsIgnoreCase("time")){
                try{
                    SimpleDateFormat timeformater = new SimpleDateFormat(
                            "hh:mm:ss",WDResourceHandler
                                    .getCurrentSessionLocale());
                    o2 = timeformater.parse(filtervalue);
                } catch (ParseException ex){
                    o2 = filtervalue;
                }
            }
            if (attributetype.equalsIgnoreCase("boolean")){
                if (sign.equalsIgnoreCase(NE)){
                    o2 = new Boolean(true);
                } else{
                    o2 = new Boolean(true);
                }
            }
            return o2;
        } catch (Exception ex){
            return filtervalue;
        }
    }
    /* compare object according the filter */
    private boolean evaluateFilteredAttributeValue(HashMap filter,
            String attributetype,Comparator comparator,Object o1){
        boolean accept = true;
        try{
            if (filter == null || filter.size() == 0)
                return accept;
            if (attributetype == null || attributetype.trim().length() == 0)
                return accept;
            Object o2 = filter.get("value1");
            if (filter.get("sign") == null){
                if (o2 != null){
                    String value1 = filter.get("value1").toString();
                    o2 = generateObject(value1,attributetype,null);
                }
                int compared = comparator.compare(o1,o2);
                if (!attributetype.equalsIgnoreCase("string")){
                    if (compared == 0){
                        return accept;
                    } else{
                        return (!accept);
                    }
                } else{
                    if (compared >= 0){
                        return accept;
                    } else{
                        return (!accept);
```

```
                }
            }
        }
        String sign = filter.get("sign").toString();
        String place = filter.get("place").toString();
        if (sign.equalsIgnoreCase(EQ) && place.equalsIgnoreCase("S")) {
            if (o2 != null) {
                String value1 = filter.get("value1").toString();
                o2 = generateObject(value1,attributetype,sign);
            } else {
                if (attributetype.equalsIgnoreCase("boolean")) {
                    o2 = generateObject("",attributetype,sign);
                }
                if (attributetype.equalsIgnoreCase("string") && o1 != null
                        &&o1.toString().trim().length() == 0) {
                    o2 = generateObject("",attributetype,sign);
                }
            }
            int compared = comparator.compare(o1,o2);
            if (! attributetype.equalsIgnoreCase("string")) {
                if (compared == 0) {
                    return accept;
                } else {
                    return (! accept);
                }
            } else {
                if (compared >= 0) {
                    return accept;
                } else {
                    return (! accept);
                }
            }
        }
        if (sign.equalsIgnoreCase(NE) && place.equalsIgnoreCase("S")) {
            if (o2 != null) {
                String value1 = filter.get("value1").toString();
                o2 = generateObject(value1,attributetype,sign);
            } else {
                if (attributetype.equalsIgnoreCase("boolean")) {
                    o2 = generateObject("",attributetype,sign);
                }
                if (attributetype.equalsIgnoreCase("string") && o1 != null
                        &&o1.toString().trim().length() == 0) {
                    o2 = generateObject("",attributetype,sign);
                }
            }
            int compared = comparator.compare(o1,o2);
            if (! attributetype.equalsIgnoreCase("string")) {
                if (compared == 0) {
                    return ! accept;
                } else {
                    return (accept);
                }
            } else {
                if (compared >= 0) {
```

```
                return ! accept;
            } else{
                return (accept);
            }
        }
    }
    if (sign.equalsIgnoreCase(R) && place.equalsIgnoreCase("S")){
        if (o2 ! = null){
            String value1 = filter.get("value1").toString();
            o2 = generateObject(value1,attributetype,sign);
        }
        int compared = comparator.compare(o1,o2);
        if (compared > 0){
            return ! accept;
        } else{
            return (accept);
        }
    }
    if (sign.equalsIgnoreCase(R) && place.equalsIgnoreCase("E")){
        if (o2 ! = null){
            String value1 = filter.get("value1").toString();
            o2 = generateObject(value1,attributetype,sign);
        }
        int compared = comparator.compare(o1,o2);
        if (compared > =0){
            return accept;
        } else{
            return (! accept);
        }
    }
    Object o3 = filter.get("value2").toString();
    if (sign.equalsIgnoreCase(R) && place.equalsIgnoreCase("M")){
        boolean accept1 = true;
        boolean accept2 = true;
        if (o2 ! = null){
            String value1 = filter.get("value1").toString();
            o2 = generateObject(value1,attributetype,sign);
        }
        if (o3 ! = null){
            String value1 = filter.get("value2").toString();
            o3 = generateObject(value1,attributetype,sign);
        }
        int compared = comparator.compare(o1,o2);
        if (compared > =0){
            accept1 = accept;
        } else{
            accept1 = ! accept;
        }
        int compared2 = comparator.compare(o1,o3);
        if (compared2 > 0){
            accept2 = ! accept;
        } else{
            accept2 = accept;
        }
        if (accept1 && accept2){
```

```
                    return accept;
                } else{
                    return ! accept;
                }
            }
            return accept;
        } catch (Exception ex){
            return accept;
        }
    }
    /* parse the filter */
    private HashMap detectFilterSign(String value){
        HashMap ar = new HashMap();
        String val1 = null;
        String val2 = null;
        String sign = null;
        String place = null;
        if (null == value){
            ar.put("value1",value);
            ar.put("sign",null);
            return ar;
        }
        if (! value.startsWith(EQ) & ! value.startsWith(NE)
                &value.indexOf(R) < 0){
            ar.put("value1",value.trim());
            ar.put("sign",null);
            return ar;
        }
        if (value.startsWith(EQ)){
            if (value.trim().length() > 1){
                val1 = value.substring(1);
            } else{
                val1 = "";
                val1 = null;
            }
            sign = EQ;
            place = "S";
            ar.put("value1",val1);
            ar.put("sign",EQ);
            ar.put("place",place);
            return ar;
        }
        if (value.startsWith(NE)){
            if (value.trim().length() > 1){
                val1 = value.substring(1);
            } else{
                val1 = "";
                val1 = null;
            }
            sign = NE;
            place = "S";
            ar.put("value1",val1);
            ar.put("sign",NE);
            ar.put("place",place);
            return ar;
```

```
        }
        if (value.startsWith(R)) {
            if (value.trim().length() > 1) {
                val1 = value.substring(1);
            } else {
                val1 = "";
                val1 = null;
            }
            sign = R;
            place = "S";
            ar.put("value1",val1);
            ar.put("sign",R);
            ar.put("place",place);
            return ar;
        }
        if (value.endsWith(R)) {
            if (value.trim().length() > 1) {
                val1 = value.substring(0,value.length() - 1);
            } else {
                val1 = "";
                val1 = null;
            }
            sign = R;
            place = "E";
            ar.put("value1",val1);
            ar.put("sign",R);
            ar.put("place",place);
            return ar;
        }
        if (value.indexOf(R) > 0) {
            sign = R;
            place = "M";
            val1 = value.substring(0,value.indexOf(R));
            val2 = value.substring(value.indexOf(R) + 1);
            ar.put("value1",val1);
            ar.put("value2",val2);
            ar.put("sign",R);
            ar.put("place",place);
            return ar;
        }
        if (ar.size() ==0)
            ar.put("value1",value);
        return ar;
    }
    private static final String bindingOfPrimaryProperty(
            IWDTableCellEditor editor) {
        return editor instanceof IWDViewElement ?
bindingOfPrimaryProperty((IWDViewElement) editor)
                : null;
    }
    private static final String bindingOfPrimaryProperty(IWDViewElement element) {
        if (element instanceof IWDAbstractDropDownByIndex)
            return ((IWDAbstractDropDownByIndex) element).bindingOfTexts();
        if (element instanceof IWDAbstractDropDownByKey)
            return ((IWDAbstractDropDownByKey)
```

```
element).bindingOfSelectedKey();
        if (element instanceof IWDAbstractInputField)
            return ((IWDAbstractInputField) element).bindingOfValue();
        if (element instanceof IWDCaption)
            return ((IWDCaption) element).bindingOfText();
        if (element instanceof IWDCheckBox)
            return ((IWDCheckBox) element).bindingOfChecked();
        if (element instanceof IWDLink)
            return ((IWDLink) element).bindingOfText();
        if (element instanceof IWDProgressIndicator)
            return ((IWDProgressIndicator) element).bindingOfPercentValue();
        if (element instanceof IWDRadioButton)
            return ((IWDRadioButton) element).bindingOfSelectedKey();
        if (element instanceof IWDTextEdit)
            return ((IWDTextEdit) element).bindingOfValue();
        if (element instanceof IWDTextView)
            return ((IWDTextView) element).bindingOfText();
        return null;
    }
    private static final Comparator DEFAULT = new Comparator() {
        public int compare(Object o1, Object o2) {
            if (o1 == null && o2 == null) {
                return 0;
            }
            if (o1 == null) {
                return -1;
            }
            if (o2 == null) {
                return -1;
            }
            if (o1 instanceof Boolean && o2 instanceof Boolean) {
                return o1.toString().compareTo(o2.toString());
            }
            if (o1 instanceof Date && o2 instanceof Date) {
                return ((java.util.Date) o1).compareTo((java.util.Date) o2);
            }
            if (o1 instanceof String && o2 instanceof String) {
                Collator collate = Collator.getInstance(WDResourceHandler
                        .getCurrentSessionLocale());
                return o1.toString().toUpperCase(
                        WDResourceHandler.getCurrentSessionLocale()).indexOf(
                        o2.toString().toUpperCase(
                                WDResourceHandler.getCurrentSessionLocale()));
            }
            return ((Comparable) o1).compareTo((Comparable) o2);
        }
    };
    private Hashtable filterForColumn = new Hashtable();
    private IWDTable table = null;
    private Hashtable attributeSource = null;
    private Hashtable sourceNodeHash = null;
    private Hashtable hashIcon = null;
    private IWDNode sourceNode = null;
    private IWDNode targetNode = null;
    String EQ = " = ";
```

```
        String NE = "#";
        String R = " ~ ";
        public final class NodeElementByAttributeComparator{
            public NodeElementByAttributeComparator(String attributeInfo,
                    IWDTableColumn column,Comparator comparator){
                this(attributeInfo,column,comparator,false,new ArrayList());
            }
            private NodeElementByAttributeComparator(String attributeInfo,
                    IWDTableColumn column,Comparator comparator,
                    boolean compareAsText,Collection subNodes){
                if (attributeInfo == null)
                    throw new IllegalArgumentException(
                            "attributeInfo must not be null");
                if (column == null)
                    throw new IllegalArgumentException("Columnmust not be null");
                if (comparator == null)
                    comparator = DEFAULT;
                this.attributeInfo = attributeInfo;
                this.column = column;
                this.comparator = comparator;
                this.compareAsText = compareAsText;
                this.subNodes = subNodes;
            }
            private final Comparator comparator;
            private final boolean compareAsText;
            private IWDTableColumn column;
            private Collection subNodes;
            private String attributeInfo;
        }
    }
```

进入文件导航器“Navigator”新建一个文件夹“util”，并将之前创建的两个工具类复制到对应目录中，如图 4-176 所示。

（1）过滤功能

1）在视图上下文中定义过滤变量，如图 4-177 所示。

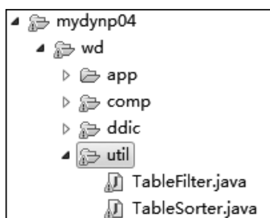


图 4-176　新建“util”文件夹

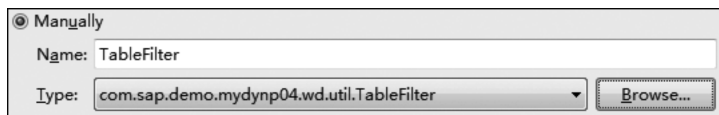


图 4-177　在上下文中添加特性

2）定义过滤用结点，过滤结点集合属性为“1‥1”，并且结构需与源数据结点保持一致，如图 4-178 所示。

3）定义源结点，源结点集合属性为“0‥n”，源结点与过滤后结点（实际业务数据结点）结构一致，如图 4-179 所示。

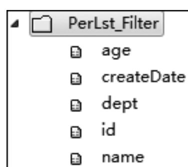


图 4-178　特性清单 1

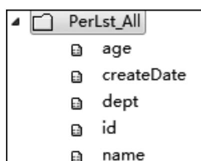


图 4-179　特性清单 2

4）创建过滤功能对应的事件，如图 4-180 所示。



图 4-180　修改事件属性

5）进入 Java 编辑器，在方法中添加以下过滤代码：

```
wdContext. currentContextElement( ). getTableFilter( ). filter(
wdContext. nodePerLst_All( ) , wdContext. nodePerLst( ) ) ;
wdDoModifyView 方法中添加初始化方法
if( firstTime) {
//获取 UI Table Element
table = ( IWDTable)  view. getElement( "Table_0" ) ;
//初始化 TableFilter 类
wdContext. currentContextElement( ). setTableFilter(
new TableFilter( table, wdThis. wdGetFilterAction( ) ,
wdContext. nodePerLst_All( ) , new Hashtable( ) ) ) ;
//初始化过滤,默认情况下过滤条件为空,Table 显示源数据
wdThis. onActionFilter( null) ;
}
```

6）选中需要设置的过滤列，如图 4-181 所示。

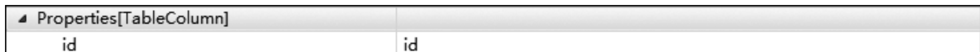


图 4-181　打开对应列的属性

7）filterValue 绑定过滤结点对应属性，如图 4-182 所示。



图 4-182　修改属性

8）设计页面显示过滤输入框，如图 4-183 所示。

9）用同样的方式设置其他列，如图 4-184 所示。



图 4-183　预览效果

人员信息列表

人员编号	姓名	年龄	部门	创建日期
@id	@name		@dept	
@id	@name	@age		@createDate
@id	@name	@age		@createDate
@id	@name	@age		@createDate

图 4-184　设置后预览效果

有的应用可能在设置 filterValue 后，输入框仍然只读，此时应该注意查看过滤结点 Initialize Lead Selection 属性。前面讲过，对于集合属性为 1‥1 的结点，可以当作单个实体来

看待，Initialize Lead Selection 为 false，那么结点 currentElement 为 null，输入框只读。另外，通过图 4-184 可以看到，只对类型为 string 的列设置了 filterValue，由于提供的标准过滤类目前只考虑了字符的情况，因此对于数字、日期等类型可以考虑通过类型转换的方式来实现。

修改初始化数据代码。当 Table 实现了过滤之后，Table 绑定的结点变为目的结点，即 Table 显示的数据一直是执行过滤 Action 后的数据。因此初始化数据时就变成了对源结点的初始化。

wdDoInit( )方法中修改为以下代码：

```
IPerLst_AllNode nodePerLst = wdContext. nodePerLst_All( ) ;                //获取源结点对象索引
//初始化两条数据
java. sql. Date date = new java. sql. Date( System. currentTimeMillis( ) ) ;
IPerLst_AllElement createAndAddPerLstElement = nodePerLst
. createAndAddPerLst_AllElement( ) ;
createAndAddPerLstElement. setId( "001" ) ;
createAndAddPerLstElement. setName( "李 1" ) ;
createAndAddPerLstElement. setAge(30) ;
createAndAddPerLstElement. setDept( "01" ) ;
createAndAddPerLstElement. setCreateDate( date) ;
createAndAddPerLstElement = nodePerLst. createAndAddPerLst_AllElement( ) ;
createAndAddPerLstElement. setId( "002" ) ;
createAndAddPerLstElement. setName( "李 2" ) ;
createAndAddPerLstElement. setAge(31) ;
createAndAddPerLstElement. setDept( "02" ) ;
createAndAddPerLstElement. setCreateDate( date) ;
```

总结：过滤的原理是通过用户输入的过滤条件从源数据中筛选出符合条件的数据，Table绑定的数据结点为目的结点。初始化时，直接往源结点中初始化数据，并且此时没有任何过滤条件，仍然执行过滤 Action，程序默认将源结点数据全部复制到目的结点，当用户执行过滤 Action 时，标准过滤类会通过过滤结点获取过滤条件，并从源结点中做过滤，过滤后数据赋值到目的结点，即表格显示的数据。这就是整个过滤的过程，所以在 Table 过滤的简单封装中，也可以围绕这个流程去实现。

10）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-185 所示。

人员信息列表

人员编号	姓名	年龄	部门
001	李1	30	人事部
002	李2	31	网络部

图 4-185　运行效果 1

11）输入过滤值，按〈Enter〉键，如图 4-186 所示。

人员信息列表

人员编号	姓名	年龄	部门
001			
001	李1	30	人事部

图 4-186　运行效果 2

（2）排序功能

1）排序比过滤简单，初始化标准排序类，并创建排序动作后便可使用。首先在上下文中定义排序特性，如图 4–187 所示。

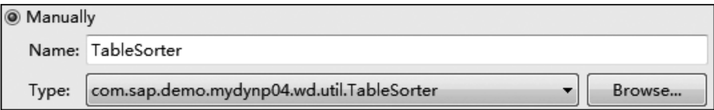


图 4–187　添加上下文特性

2）创建排序的动作，如图 4–188 所示。



图 4–188　修改事件属性

3）在方法中实现逻辑，代码如下：

```
wdContext. currentContextElement( ). getTableSorter( ). sort( wdEvent, wdContext. nodePerLst( ) );
wdDoModifyView 中添加初始化 TableSorter 类方法。
if( firstTime) {
wdContext. currentContextElement( ). setTableSorter( new
TableSorter( table, wdThis. wdGetSortAction( ) , null) ) ;
}
```

4）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4–189 所示。

人员信息列表

人员编号	姓名	年龄	部门	创建日期
001	李1	30	人事部	2013-8-27
002	李2	31	网络部	2013-8-27

图 4–189　运行效果

5）单击列头即可进行排序，如图 4–190 所示。

人员信息列表

人员编号	姓名	年龄	部门
001	李1	30	人事部
002	李2	31	网络部

图 4–190　排序后的效果

4.3.4　树

其实在 Web Dynpro 的上下文定义中一直使用的就是一个树状结构，而 SAP 为了将整个上下文结构进行可视化的展示，定义了 Web Dynpro 中的树。开发人员在进行设计开发时，如果无法确定树的层级（深度），则可以在上下文定义时使用“循环结点”，一般循环结点使用的就是它父结点的对象；如果能够确定树的层级，则无须使用循环结点，使用一般类型的结点即可。

它包含以下一系列属性。

- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3

种类型。

- contextMenuId：设置上下文菜单的唯一标识。
- dataSource：树所绑定的上下文结点数据源。
- defaultItemIconSource：控制显示在叶结点的图标路径。
- defaultNodeIconSource：控制显示在结点的图标路径。
- enabled：控制元素是否可用。
- minHeight：控制树元素的最小高度。
- rootText：控制根结点的文本描述。
- rootVisible：控制根结点是否可见。
- selectionChangeBehaviour：设置触发事件时，是否自动切换父结点的首选项（lead selection），有自动或手动两个选项。
- titleVisible：控制树的标题是否可见。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- width：控制元素的显示宽度。

树结点的属性如下。

- dataSource：绑定的数据源。
- hasChildren：是否含有下级结点。
- ignoreAction：是否忽略选择触发的 Action。
- text：Tree 结点的描述。

开发实例：

1）首先在视图上下文中定义树的数据源结点和特性，如图 4-191 所示。

①“TreeNode”的集合属性为“0··n”，是否初始化首选项“Initialize Lead Selection”为“false”。

② 特性“id”和“name”的类型为字符串型，用于存放代码与名称。

③ 特性“hasChildren”的类型为布尔型，控制是否含有下级结点。

④ 特性“ignoreAction”的类型为布尔型，控制是否忽略选择的 Action。

⑤ 特性“isExpanded”的类型为布尔型，控制是否能够展开下级结点。

2）右击“TreeNode”，添加一个循环结点（Recursion Node），输入循环结点的名称，如图 4-192 所示。

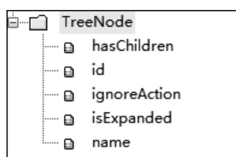


图 4-191 树元素对应的上下文结构

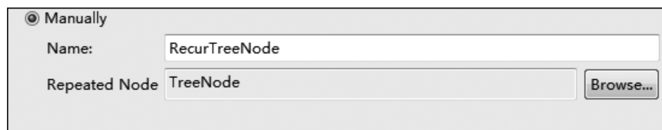


图 4-192 创建循环结点

3）创建完成后，保存所有修改，如图 4-193 所示。

4）右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“Group”，输入 ID 名称，单击“OK”按钮。将组的宽度定义为“100%”，将“layout”属性修改

为“GridLayout”，将“Header”修改为“组织体系”。在组中添加两个透明容器，将“LayoutData”中的“Width”分别设置为“30%”和“70%”，如图4-194所示。

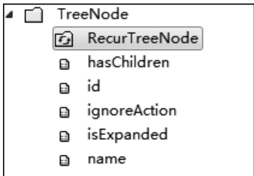


图4-193　创建后的效果

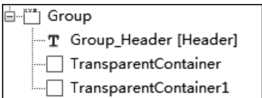


图4-194　元素清单

5）右击“TransparentComtainer”透明容器，选择插入一个子对象，选择“Tree”，输入ID名称，单击“OK”按钮，如图4-195所示。

6）右击“Tree”，选择“InsertNodeType”→“TreeNodeType”，插入一个结点，如图4-196所示。

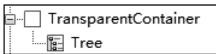


图4-195　创建后的效果

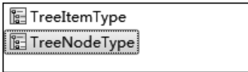


图4-196　选择树结点类型

7）完成界面元素定义后，在元素清单中可以看到新的界面结构，如图4-197所示。

8）选中“Tree”，将“Datasource”数据源属性与上下文中的“TreeNode”结点进行绑定，如图4-198所示。

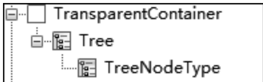


图4-197　添加结点后的效果



图4-198　选择对应的上下文进行绑定

9）修改“rootText”属性为“组织架构”，修改“width”为“200px”，如图4-199所示。

dataSource	TreeNode
defaultItemIconSource	
defaultNodeIconSource	
enabled	true
minHeight	
rootText	组织架构
rootVisible	true
selectionChangeBehaviour	auto*
title	
titleVisible	visible*
tooltip	
visible	visible*
width	200px

图4-199　修改属性

10）接着修改树的结点“TreeNodeType”。将“dataSource”属性与上下文中的“TreeNode”进行绑定，其他属性参考图4-200进行修改。由于使用的是循环结点，因此视图中树元素的数据源和叶结点的数据源使用的是同一个上下文对象。

11）完成所有配置后，在视图中能看到界面的预览图，如图4-201所示。

12）接着在上下文中新建一个结点和两个特性，用于显示明细数据信息。模拟选中单

条项目时明细显示，结点“Detail”的结合属性设置为“1··1”，如图 4-202 所示。

dataSource	TreeNode
design	standard*
expanded	TreeNode.isExpanded
hasChildren	TreeNode.hasChildren
iconSource	
ignoreAction	TreeNode.ignoreAction
text	TreeNode.name
textDirection	inherit*
tooltip	

图 4-200　树元素的属性

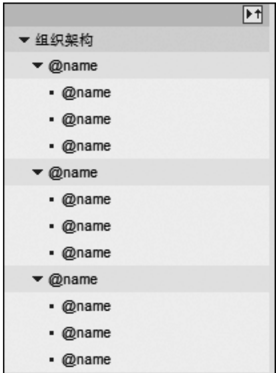


图 4-201　预览效果

13）右击“TransparentContainer1”并选择应用模板，随后选择表单类型，并使用“Detail”结点进行表单的自动生成，如图 4-203 所示。

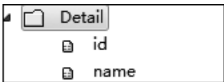


图 4-202　添加特性后的效果

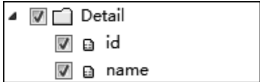


图 4-203　选择对应的上下文

14）两个特性在表单中的显示都使用输入框元素进行展现，如图 4-204 所示。

15）模板运行完毕后，自动生成表单相关元素，如图 4-205 所示。

Name	Attribute	Editor	Bound Prop...
id	id (TreeNode.id)	InputField	value
name	name (TreeNode.nam...	InputField	value

图 4-204　修改元素的类型和位置

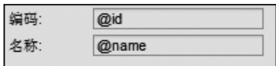


图 4-205　预览效果

16）将表单与之前的树元素一同放入组“Group”中，如图 4-206 所示。

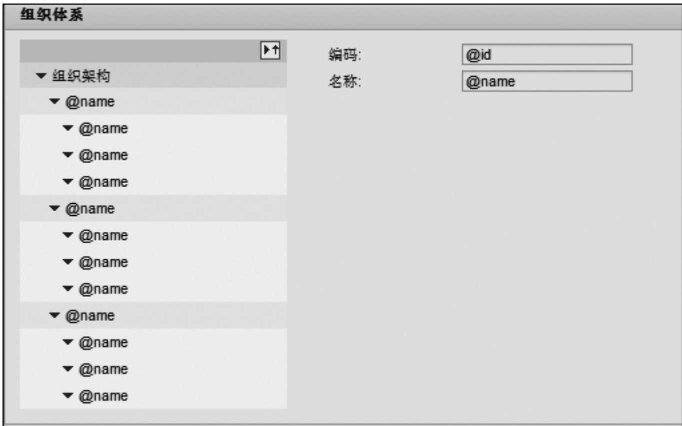


图 4-206　运行效果

17）在视图的“wdDoInit”初始化方法中添加下列测试代码：

```
//初始化 Tree
ITreeNodeNode nodeTreeNode = wdContext.nodeTreeNode();
ITreeNodeElement createAndAddTreeNodeElement = nodeTreeNode
        .createAndAddTreeNodeElement();
createAndAddTreeNodeElement.setId("OC0001");           //赋值代码
createAndAddTreeNodeElement.setName("SAPEYES01");      //赋值名称
createAndAddTreeNodeElement.setHasChildren(true);
createAndAddTreeNodeElement.setIgnoreAction(true);
createAndAddTreeNodeElement = nodeTreeNode
        .createAndAddTreeNodeElement();
createAndAddTreeNodeElement.setId("OC0002");               //赋值代码
createAndAddTreeNodeElement.setName("SAPEYES02");          //赋值名称
createAndAddTreeNodeElement.setHasChildren(true);
createAndAddTreeNodeElement.setIgnoreAction(true);
createAndAddTreeNodeElement = nodeTreeNode
        .createAndAddTreeNodeElement();
createAndAddTreeNodeElement.setId("OC0003");               //赋值代码
createAndAddTreeNodeElement.setName("SAPEYES03");          //赋值名称
createAndAddTreeNodeElement.setHasChildren(true);
createAndAddTreeNodeElement.setIgnoreAction(true);
```

18）定义树结点的标准事件“onLoadChildren”，定义新动作，如图 4-207 所示。

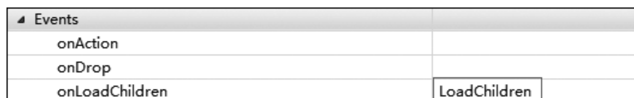


图 4-207　修改事件属性

19）添加事件处理器中的方法参数，如图 4-208 所示。

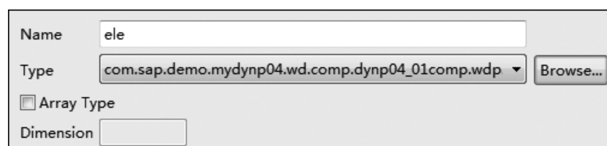


图 4-208　添加参数

20）参数类型选择“TreeNodeElement”生成的本地类，如图 4-209 所示。

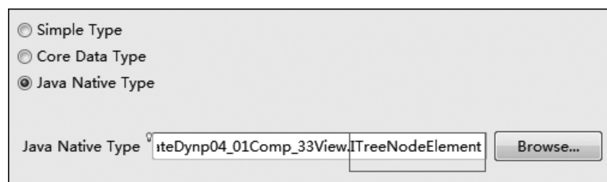


图 4-209　选择对应类型

21）复制参数至参数中，如图 4-210 所示。



图 4-210　添加后的效果

22）定义元素的参数映射，如图 4-211 所示。将 TreeNodeType onLoadChildren 与自定义的 Action 参数做关联。

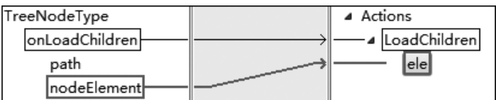


图 4-211　进行参数映射

23）进入 Event Handler 实现，通过循环结点编写加载下级结点逻辑：

```
//获取循环结点
    ITreeNodeNode nodeRecurTreeNode = ele. nodeRecurTreeNode( ) ;
    String id = ele. getId( ) ;
    //根据选中的当前结点判断是否构造下级树
    if( id! = null && id. contains( "P" ) ) {
        //选择的结点类型为人员时不再填充下级结点
        ele. setHasChildren( false) ;
        ele. setIsExpanded( false) ;
    } else if( id! = null && id. contains( "OC" ) ) {
        //选择的结点类型为公司时,填充下级部门结点
        ITreeNodeElement createAndAddTreeNodeElement = nodeRecurTreeNode
            . createAndAddTreeNodeElement( ) ;
        createAndAddTreeNodeElement. setId( "OD0001" ) ;              //赋值代码
        createAndAddTreeNodeElement. setName( "部门 1" ) ;            //赋值名称
        createAndAddTreeNodeElement. setHasChildren( true) ;
        createAndAddTreeNodeElement. setIgnoreAction( true) ;
        createAndAddTreeNodeElement = nodeRecurTreeNode
            . createAndAddTreeNodeElement( ) ;
        createAndAddTreeNodeElement. setId( "OD0002" ) ;              //赋值代码
        createAndAddTreeNodeElement. setName( "部门 2" ) ;            //赋值名称
        createAndAddTreeNodeElement. setHasChildren( true) ;
        createAndAddTreeNodeElement. setIgnoreAction( true) ;
    } else{
        //其余情况,即当选择的结点类型为部门时,填充下级人员结点
        ITreeNodeElement createAndAddTreeNodeElement = nodeRecurTreeNode
            . createAndAddTreeNodeElement( ) ;
        createAndAddTreeNodeElement. setId( "P0001" ) ;               //赋值代码
        createAndAddTreeNodeElement. setName( "人员 1" ) ;            //赋值名称
        createAndAddTreeNodeElement = nodeRecurTreeNode
            . createAndAddTreeNodeElement( ) ;
        createAndAddTreeNodeElement. setId( "P0002" ) ;               //赋值代码
        createAndAddTreeNodeElement. setName( "人员 2" ) ;            //赋值名称
    }
```

24）定义“onAction”事件，即在选择当前结点时触发，如图 4-212 所示。

25）按照上面的方法添加参数并做元素的参数映射，如图 4-213 所示。

Events	
onAction	SelectEle
onDrop	
onLoadChildren	LoadChildren

图 4-212　修改事件属性

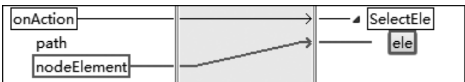


图 4-213　进行参数映射

26）添加动作的实现逻辑：

```
//直接调用 CopyService API,赋值明细
WDCopyService.copyCorresponding(ele, wdContext.currentDetailElement());
```

27）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-214 所示。

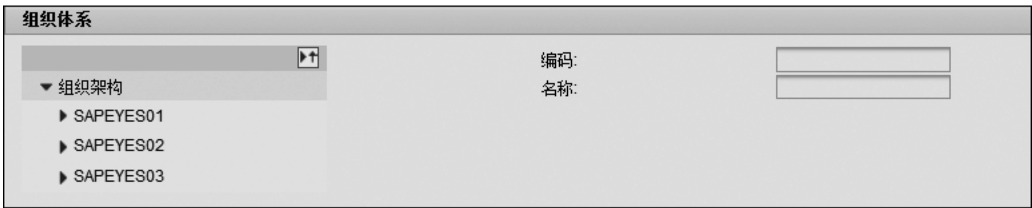


图 4-214　运行效果

28）展开公司结点，并选中一条数据，在右侧可以看到结点的明细数据，如图 4-215 所示。

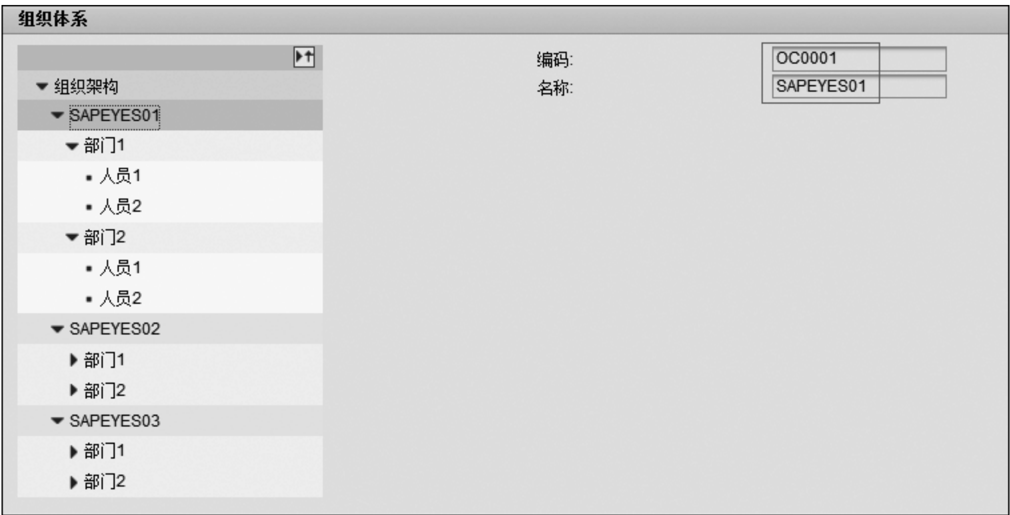


图 4-215　展开后的效果

4.3.5　表格树

表格元素通过插入行编排“RowArrangement”，可以实现树元素的功能，此时的树也会根据实际结点级别分为静态树与动态树。

开发实例：

1）将表格通过“TreeByNestingTableColumn”的方式实现动态多级结构。右击表格“Table”，选择“Insert RowArrangement”，在类型中选择“TreeByNestingTableColumn”，如图 4-216所示。

2）右击“RowArrangement”，依次选择“Insert Header”和“Insert TextView”，插入标题和文本显示区，如图 4-217 所示。



图 4-216　添加结点后效果

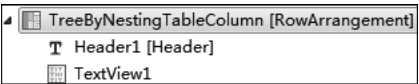


图 4-217　元素清单

3）随后，在视图中创建对应的上下文结构，包含了“HRInfoLst”结点和两个特性，如图 4-218 所示。

4）右击“HRInfoLst”，选择新建一个循环结点，如图 4-219 所示。



图 4-218　特性清单

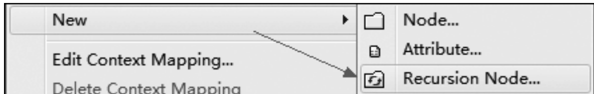


图 4-219　添加循环结点的菜单选择

5）输入循环结点名称“RepHrs”，单击“OK”按钮，如图 4-220 所示。

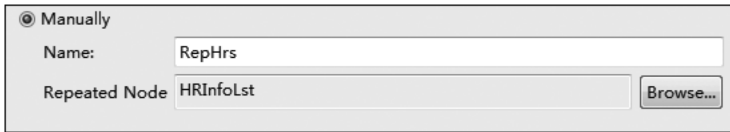


图 4-220　创建循环结点

6）绑定结点属性至元素的对应属性，其中“TreeByNestingTableColumn”列用于显示“id”字段，另外添加一列显示人员名称，如图 4-221 所示。

7）添加控制，“TreeByNesting”的结点含有以下 3 个控制树级别的属性。

- “childrenLoaded”：是否加载下一级结点。
- “expended”：是否树结点可展开。
- “isLeaf”：标明该结点是否是最后一级。

8）绑定上下文后，创建一个默认动作“LoadChildren”用于加载树的子结点，如图 4-222 所示。

人员信息架构

	名称
@id	@name
@id	@name
@id	@name

图 4-221

onLoadChildren	LoadChildren

图 4-222　修改事件属性

9）在创建动作时添加两个参数，分别为字符串类型和上下文的结点元素类型，用于记录子结点的路径以及子结点的对象，如图 4-223 所示。

Details | Parameters

Name	Type
path	string
ele	com.sap.tc.webdynpro.progmodel.api.IWDNodeElement

图 4-223　添加参数定义

10）保存所有修改，右击“Outline”清单中的表格列，选择“Parameter Mapping”（如果在创建动作时不添加对应参数，则无法进行映射），将表格列中的树与创建的动作中的参

数进行映射，如图 4-224 所示。

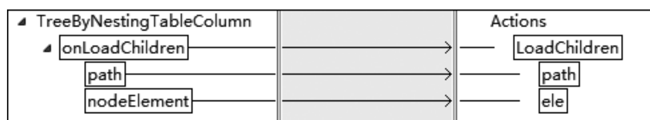


图 4-224　进行参数映射

11）接着编写动作的代码逻辑，在完成参数映射后可以看到元素在“onLoadChildren”动作的 Java 编辑器中自动传递了“path”与“nodeElement”两个变量，分别代表当前层级路径与当前层级的元素。通过两个参数可以控制不同的级别在展开树结构时显示不同的数据。参考代码如下：

```
IHRInfoLstElement element = (IHRInfoLstElement) ele;    //转换 ele
IHRInfoLstNode nodeRepHrs = element. nodeRepHrs( );  //获取子结点
nodeRepHrs. invalidate( );                              //清空数据
int level = path. split( "RepHrs" ). length;        //根据含有循环结点结构名的次数判断当前级别
if(1 == level){
//第一级展开,显示所有公司
IHRInfoLstElementcreateAndAddHRInfoLstElement  =  nodeRepHrs. createAndAddHRInfoLstElement
( );
createAndAddHRInfoLstElement. setId( "0001" );
createAndAddHRInfoLstElement. setName( "公司 1" );
createAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( );
createAndAddHRInfoLstElement. setId( "0002" );
createAndAddHRInfoLstElement. setName( "公司 2" );
createAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( );
createAndAddHRInfoLstElement. setId( "0003" );
createAndAddHRInfoLstElement. setName( "公司 3" );
}else if(2 == level){
//第二级展开,显示该公司下的部门
String id = element. getId( );              //根据 Parameter Mapping 传递的 ele 取得当前公司代码
String name = element. getName( );          //根据 Parameter Mapping 传递的 ele 取得当前公司名称
IHRInfoLstElementcreateAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( );
                                                                                    //loadChildren
//赋值 Id,Name
createAndAddHRInfoLstElement. setId( id + "01" );
createAndAddHRInfoLstElement. setName( "部门 1" );
createAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( );
createAndAddHRInfoLstElement. setId( id + "02" );
createAndAddHRInfoLstElement. setName( "部门 2" );
createAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( );
createAndAddHRInfoLstElement. setId( id + "03" );
createAndAddHRInfoLstElement. setName( "部门 3" );

}else if(3 == level){
//第三级展开,显示该部门下的人员
String id = element. getId( );              //根据 Parameter Mapping 传递的 ele 取得当前部门代码
String name = element. getName( );          //根据 Parameter Mapping 传递的 ele 取得当前部门名称
IHRInfoLstElementcreateAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( );
                                                              //loadChildren
//赋值 Id,Name
createAndAddHRInfoLstElement. setId( id + "01" );
```

```
createAndAddHRInfoLstElement. setName( " 人员 1" ) ;
createAndAddHRInfoLstElement. setExpended( false) ;               //设置当前级不可展开
createAndAddHRInfoLstElement. setChildrenLoaded( false) ;         //设置不可加载子结点
createAndAddHRInfoLstElement. setIsLeaf( true) ;                  //设置该级为末级
createAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( ) ;
createAndAddHRInfoLstElement. setId( id + "02" ) ;
createAndAddHRInfoLstElement. setName( " 人员 2" ) ;
createAndAddHRInfoLstElement. setExpended( false) ;
createAndAddHRInfoLstElement. setChildrenLoaded( false) ;
createAndAddHRInfoLstElement. setIsLeaf( true) ;
createAndAddHRInfoLstElement = nodeRepHrs. createAndAddHRInfoLstElement( ) ;
createAndAddHRInfoLstElement. setId( id + "03" ) ;
createAndAddHRInfoLstElement. setName( " 人员 3" ) ;
createAndAddHRInfoLstElement. setExpended( false) ;
createAndAddHRInfoLstElement. setChildrenLoaded( false) ;
createAndAddHRInfoLstElement. setIsLeaf( true) ;
}
```

注：该代码是针对不同级别显示不同的子结点内容，为了快速构建树结构，这里直接判断级别后创建子结点，实际封装时可以根据“path”与“ele”参数动态构造下级结点。

12）在视图的“wdDoInit”初始化方法中添加如下代码：

```
IHRInfoLstNode nodeHRInfoLst = wdContext. nodeHRInfoLst( ) ;
IHRInfoLstElementcreateAndAddHRInfoLstElement = nodeHRInfoLst. createAndAddHRInfoLstElement
( ) ;                    //创建一条根记录
createAndAddHRInfoLstElement. setId( "1001" ) ;
createAndAddHRInfoLstElement. setName( "集团" ) ;
```

13）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-225 所示。

人员信息架构

	名称
▾ 1001	集团
▾ 0001	公司1
▾ 000101	部门1
▪ 00010101	人员1
▪ 00010102	人员2
▪ 00010103	人员3
▾ 000102	部门2
▪ 00010201	人员1
▪ 00010202	人员2
▪ 00010203	人员3
▸ 000103	部门3
▸ 0002	公司2
▸ 0003	公司3

图 4-225　运行效果

4. 3. 6　文件上传

文件上传元素（FileUpload）由一个输入框和一个固定的“Browse”按钮组成，单击

“Browse”按钮，允许用户从本地硬盘中选择文件进行上传。

需要特别注意的是，文件上传元素的属性中并没有事件的定义，所以单击“浏览”按钮也不会触发服务器端的事件处理器，只是将客户端的文件存放到了程序的上下文中。所以文件上传元素本身只能完成一般上传需求一半的功能，并未实现存放至文件服务器中的功能。一般情况下，文件上传元素都需要额外定义一个按钮，以实现文件从 Web Dynpro 工程的上下文存放到指定服务器的功能。

它包含了下列属性。

- activateAccessKey：设置是否允许快捷键（使用“Alt”键与首字母触发默认事件）。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- enabled：控制元素是否可用。
- explanation：设置元素的解释文本，设置后元素文本下方会出现一条绿色的下画线。当鼠标悬停在文本上时，会弹出解释文本的说明。
- resource：用于控制用户上传文件所存放的上下文特性，该特性类型必须是“Resource”资源类型，必须绑定上下文。
- state：控制元素的显示状态，默认有“normal”和“required”两种状态。在选择“required”后，元素会出现一个红色星号进行提醒，但逻辑控制仍需要编写相关代码。
- text：控制文件上传的显示文本。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- width：控制元素的显示宽度。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“FileUpload”文件上传类型，输入 ID 名称，单击“OK”按钮，如图 4-226 所示。

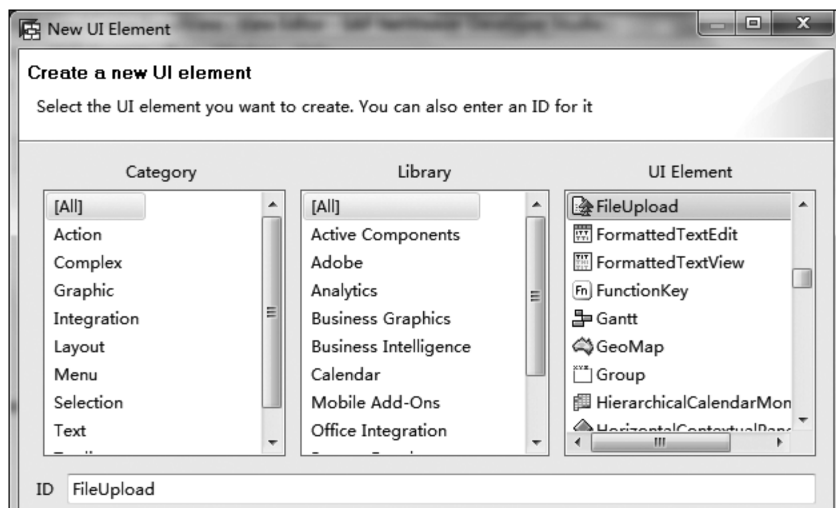


图 4-226　选择元素类型

2）进入上下文页签，创建一个新的上下文特性。右击“Context”根结点，选择“New”→“Attribute”，如图 4-227 所示。

3）在弹出的对话框中输入特性的名称“resource”，随后单击“Browse”按钮，选择特性类型，如图 4-228 所示。

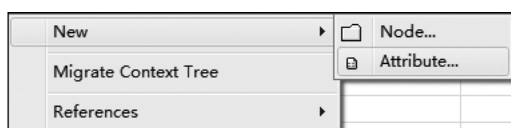


图 4-227　创建结点的菜单选择

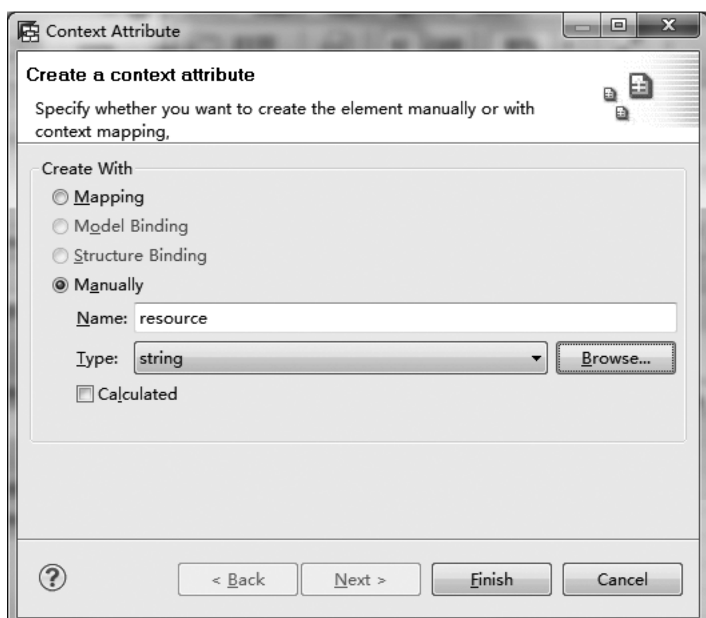


图 4-228　输入特性名称

4）在弹出的对话框的类型列表中找到“com. sap. ide. webdynpro. uielementdefinitions”包，如图 4-229 所示。

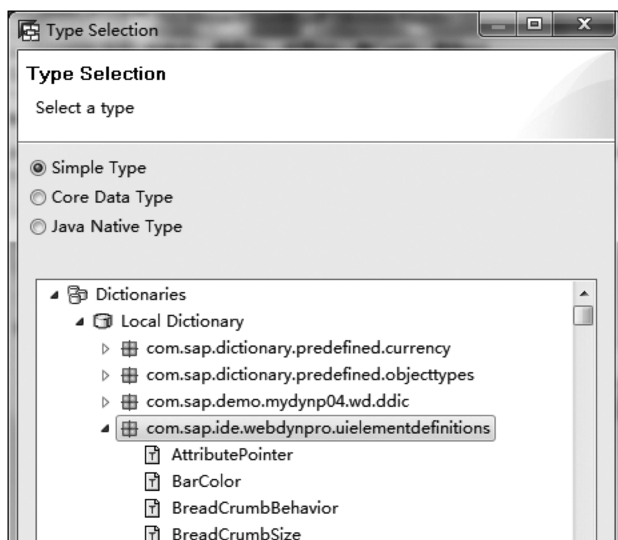


图 4-229　选择类型

5）在该包的类型列表中选中“resource”资源类型，单击“OK”按钮完成特性的类型选择，单击“Finish”按钮完成特性的创建，如图 4-230 所示。

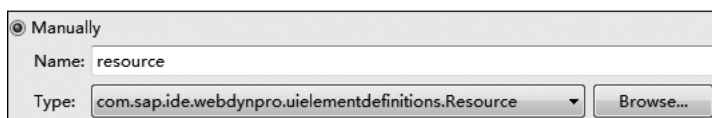


图 4-230　创建特性

6）选中视图中的文件上传元素，将创建的特性与元素属性进行绑定，如图 4-231 所示。

7）添加“解析”按钮，并创建默认动作解析文件，如图 4-232 所示。注：在实际项目实施过程中，“解析”按钮更多的是完成将文件解析并存放至如 FTP 服务器的文件目录中。

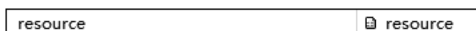


图 4-231　修改属性

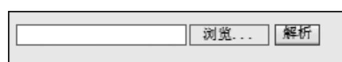


图 4-232　运行效果

8）保存所有修改后，进入 Java 编辑器，在按钮动作中编写文件处理逻辑，添加下列代码：

```
//获取 resource 属性
IWDResource resource = wdContext. currentContextElement( ). getResource( );
if( resource! = null) {
String resourceName = resource. getResourceName( );
wdComponentAPI. getMessageManager( ). reportSuccess(
" 打印 ResourceName:" + resourceName);
//以下为解析 TxT 文件
try{
InputStream read = resource. read( true);                    //resource 读取文件输入流
InputStreamReader reader = new InputStreamReader(
read, "GBK");                                                //根据文件输入流构造 Reader
BufferedReader bufferedReader = new BufferedReader( reader);
String lineTxt = null;
while( ( lineTxt = bufferedReader. readLine( ) ) ! = null) {   //逐行解析文件
wdComponentAPI. getMessageManager( ). reportSuccess(
" 文件内容:" + lineTxt);                                     //打印文件内容
}
read. close( );                                              //关闭文件输入流
} catch( IOException e) {
//TODO Auto - generated catch block
//e. printStackTrace( );
//解析文件异常
wdComponentAPI. getMessageManager( ). reportException( "解析的文件异常!" );}
} else{
//未浏览选择文件
wdComponentAPI. getMessageManager( ). reportException( "解析的文件为空!" );
}
```

9）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-233 所示。

10）单击“浏览”按钮，选择文件，随后单击“解析”按钮，如图 4-234 所示。

11）单击“解析”按钮后可以看到打印的消息，如图 4-235 所示。

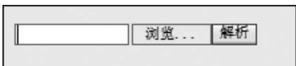


图 4-233　运行效果



图 4-234　选择浏览文件

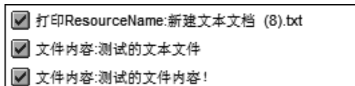


图 4-235　单击“解析”按钮后的效果

4.3.7　文件下载

文件下载元素（FileDownload）由一个标题和一个链接组成，根据不同的动作属性配置，用户可以直接打开链接对应的服务器端的文件或者转存至用户的本地硬盘中。与文件上传元素一样，文件下载元素的核心也是使用一个类型为“Resource”资源的上下文特性对文件进行处理。文件下载也同样没有事件绑定，从逻辑角度看，用户单击链接只是将文件下载元素绑定的上下文中的文件资源进行处理，所有操作都在客户端完成。文件下载元素包含下列属性。

1）activateAccessKey：设置是否允许快捷键（使用“Alt”键与首字母触发默认事件）。

2）behavior：控制用户在单击文件下载元素后的反馈动作，默认有 3 个选项：“allowSave”允许保存，在用户单击后会首先弹出一个“打开”或“保存”对话框，即允许用户直接使用默认程序打开文件或者直接另存至本地硬盘中；“auto”自动，程序会根据文件资源的类型自动进行响应，表 4-4 为部分文件类型的响应方式清单，其中最后一列的“是”与“否”表示是否会在新窗口中打开；“openInPlace”表示对应文件将使用用户客户端的默认软件在同一窗口中打开。

表 4-4　部分文件类型的响应方式清单

常量	文件扩展名	资源类型（W3C 标准）	新窗口打开
APPLET	“jar”	application/x - java	是
CSS	“css”	text/css	否
DOC	“doc”	application/vmd. ms - word application/x - msword	否
FLASH	“swf”	application/x - shockwave - flash	否
GIF_IMAGE	“gif”	image/gif	否
HTML	“html”	text/html	否
JAVA	“java”	application/x - java	否
JAVA_SCRIPT	“js”	text/js	否
JPG_IMAGE	“jpg”	image/jpeg	否
PDF	“pdf”	application/pdf	否
PNG	“png”	image/jpeg	否
PPT	“ppt”	application/vmd. ms - powerpoint application/x - mspowerpoint	否
PS	“ps”	application/postscript	否
RTF	“rtf”	application/rtf	否
SVG	“svg”	image/svg + xml	否

（续）

常量	文件后缀	资源类型（W3C 标准）	新窗口打开
TXT	“txt”	text/text	否
UNKNOWN	–	–	是
VML	“vml”	text/vml	否
WD_APPLICATION	–	–	否
XLS	“xls”	application/vmd. ms – excel application/x – msexcel	否
XML	“xml”	text/xml	否
XML_CONFIGURATION	“xml”	text/xml	是

3） contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。

4） contextMenuId：设置上下文菜单的唯一标识。

5） design：控制链接文本的显示风格，默认有标准显示和增强显示两种类型，其中增强类型即为文本加粗显示。

6）enabled：控制元素是否可用。

7） imageFirst：控制显示的链接图标是否位于文本前面，布尔型属性。

8） imageHeight：控制文件下载元素中链接图标的高度。

9） imageSource：控制文件下载元素中链接图标的来源（文件目录位置）。

10） imageWidth：控制文件下载元素中链接图标的宽度。

11） resource：用于控制用户下载文件所对应的上下文特性。该特性类型必须是“Resource”资源类型，必须绑定上下文。

12） target：控制弹出页面的具体位置，如可以使弹出页面在指定的窗口中打开。

13） text：控制文件上传的显示文本。

14） textDirection：控制链接文本的排列顺序，默认有 3 个选项：继承自父元素、从左往右和从右往左。

15） tooltip：控制提示消息的显示文本。

16） type：控制文件下载元素中链接的显示风格，默认有以下 4 种风格：

- 功能型：显示默认的链接风格，默认为蓝色带下画线文本。
- 导航型：显示为已访问过的链接颜色，文本带下画线。
- 报告型：显示默认的链接风格，文本不带下画线。
- 结果型：显示为已访问过的链接颜色，文本不带下画线。

17） visible：控制元素是否可见。

18） width：控制元素的显示宽度。

开发实例：

1） 以上一节中的文件上传元素的实例为基础，增加一个文件下载元素，能实现用户在上传文件后即能看到是否上传成功以及对应文件下载的效果。首先打开文件上传的视图（如需重新创建，请参考上一节“文件上传”中开发实例的步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“FileDownload”文件下载类型，输入 ID 名称，随后单击“OK”按钮，如图 4-236 所示。

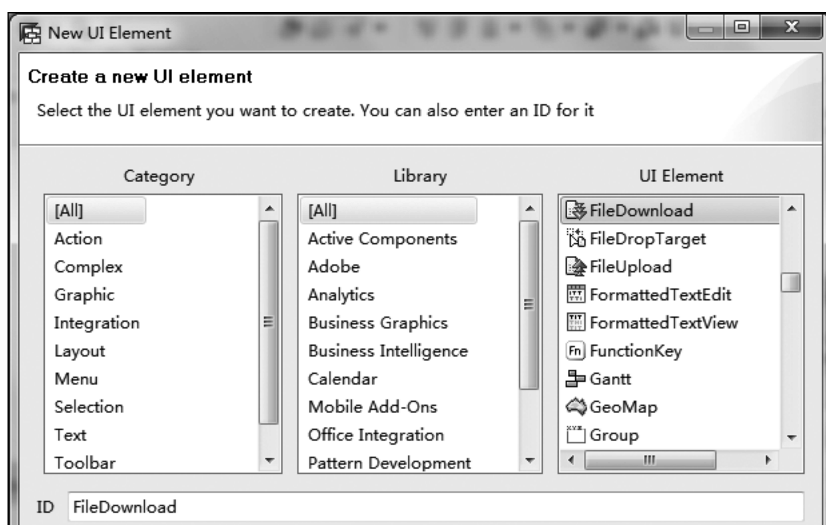


图 4-236　选择元素类型

2）此时会用到 3 个元素：文件上传、上传按钮和文件下载，如图 4-237 所示。注意，文件上传是将文件从用户客户端上传至 Web Dynpro 程序的上下文中，而上传按钮是将上下文中的文件存储至服务器端。

3）进入上下文页签，创建一个新的上下文特性，用于在文件下载元素中显示对应的文件名。右击“Context”根结点，选择“New”→“Attribute”，如图 4-238 所示。

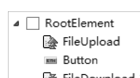


图 4-237　元素列表

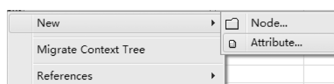


图 4-238　创建结点的菜单选择

4）在弹出的对话框中输入特性的名称“fileName”，类型使用默认的字符串类型，单击“Finish”按钮，如图 4-239 所示。

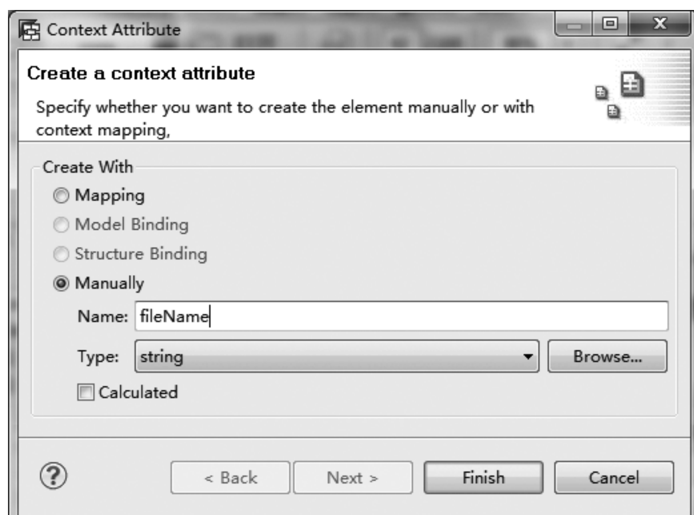


图 4-239　输入特性名称

5）保存所有修改，完成了上下文特性的创建，如图 4-240 所示。注：“resource”特性是上一小节中使用“文件上传”时已经创建的特性，类型是“com. sap. ide. webdynpro. uielementdefinitions”中的“resource”资源类型。

6）选中视图中的文件下载元素，将创建的特性与元素属性进行绑定。其中“resource”属性直接绑定文件上载元素数据源，即通过文件上传元素得到的文件资源同时也成为了文件下载元素的文件资源，文件下载的文本则显示出了上传文件的文件名。另外，将“behaviour”属性改为“allowSave”，允许用户直接操作文件，如图 4-241 所示。

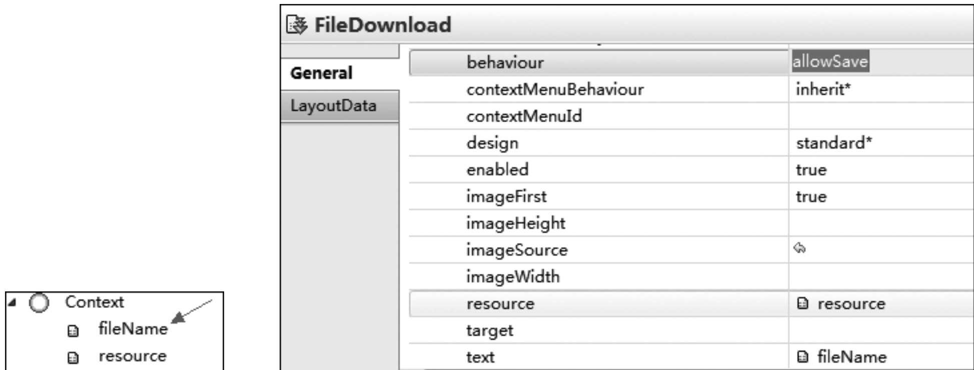


图 4-240　特性清单　　　　图 4-241　修改属性

7）在开发工具中的页面预览，如图 4-242 所示。

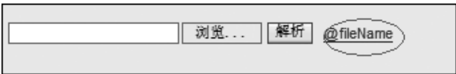


图 4-242　运行效果

8）在“解析”按钮的事件处理方法中增加下列代码，用于读取上传的文件名：

```
//这段代码需放在方法实现的最前面
IWDResource resource = wdContext. currentContextElement( ). getResource( );
if( resource! = null) {
String resourceName = resource. getResourceName( );
wdComponentAPI. getMessageManager( ). reportSuccess(
"打印 ResourceName:" + resourceName);
//赋值 fileName
wdContext. currentContextElement( ). setFileName( resourceName);
//以下为文件上传元素章节中原解析 TxT 文件代码
//…
```

9）保存所有修改，并重新编译、运行之后，即可进行上传、下载文件的测试，如图 4-243所示。

10）单击“浏览”按钮，从本地文件中选择“测试文本 . txt”，随后单击“解析”按钮，文件下载链接点亮，并显示上传文件的文件名。单击文件名链接，即可将文件另存至本地硬盘目录中，如图 4-244 所示。

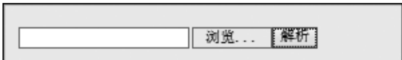


图 4-243　运行效果

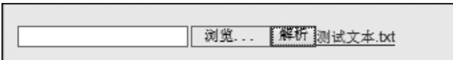


图 4-244　运行效果

注：实际使用时，上传按钮会多增加一步，将文件存放至服务器端。而在下载文件元素的初始化处理中，也同样会增加将服务器端文件读取并存放至 Web Dynpro 的上下文中的动

作。本实例中未体现这些步骤，各位读者可以在系统环境允许的条件下进行测试。

4.3.8 视图容器

在实际的项目中，尤其是跟 J2EE 实施的项目做对比，可以发现 WD4J 在设计开发过程中占据独特的优势，由于 WD4J 本身就是对 J2EE 的封装，并且是在 J2EE 基础之上的一层架构，因此其在设计、开发，特别是页面的设计开发过程中更具模板化与快速定制的特点，即开发人员可以根据后台的数据模型快速地定制符合需求的前台展现。那么除此之外，WD4J 的设计也具备较高的可重用性，对于很多页面都有的设计，可以抽象成一个界面，并且所有的页面来引用，视图容器就是实现此类功能的空间。它包含下列属性。

- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- enabled：控制元素是否可用。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。

开发实例：

1）新建一个 Web Dynpro 项目“mydynp06”，并创建对应的组件、视图和窗口，如图 4-245 所示。

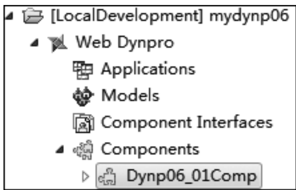


图 4-245 创建的 Web Dynpro 项目结构

2）编辑默认创建的视图，新建审批历史的数据结点并在页面生成表格，如图 4-246 所示。

审批历史

审批人	审批结果	审批意见	审批日期
@apper	@appres	@appcom	@appdate
@apper	@appres	@appcom	@appdate
@apper	@appres	@appcom	@appdate

图 4-246 预览效果

3）接着将组件添加至公共部分，右击“Dynp06_01Comp”并选择“Add to Public Part”，如图 4-247 所示。

4）在弹出的对话框中单击“New”按钮，新建一个公共部分，如图 4-248 所示。

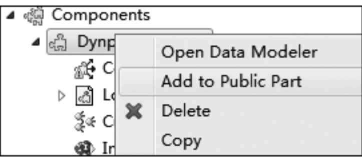


图 4-247 添加公共部分的菜单选择

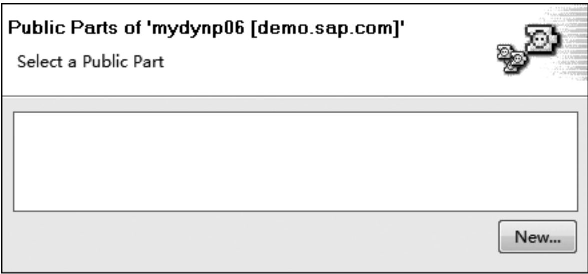


图 4-248 创建公共部分对话框

5）输入名称，单击“Finish”按钮，如图 4-249 所示。

6）创建完成公共部分后，选中该对象，随后单击“Finish”按钮，如图 4-250 所示。

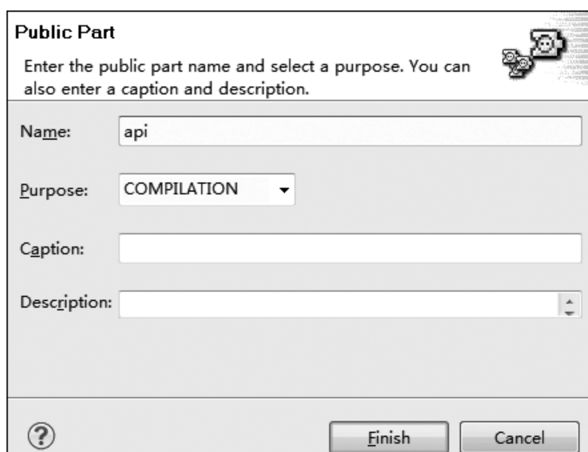


图 4-249　输入公共部分信息

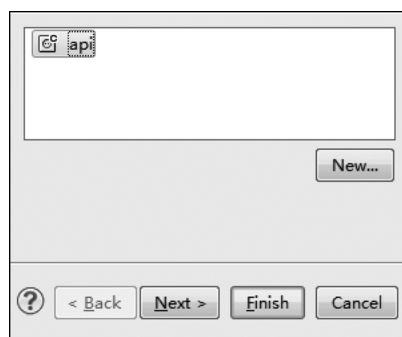


图 4-250　选择公共部分

7）切换到 DI 透视图，在“mydynp06”项目的公共部分中可以看到对应的“api”，如图 4-251 所示。

8）接着创建“mydynp05”项目到“mydynp06”的依赖。双击打开“mydynp05”后，在“Dependencies”依赖中，单击“Add”按钮，如图 4-252 所示。

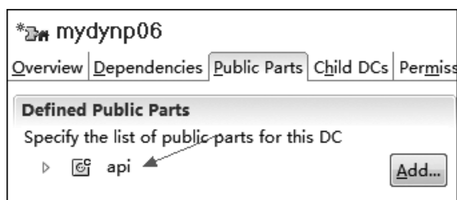


图 4-251　在 DI 透视图中示例

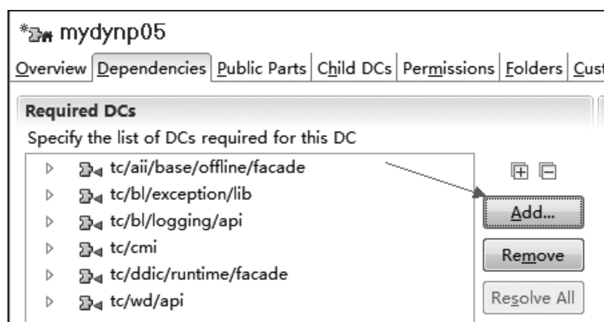


图 4-252　添加工程的依赖

9）在弹出的对话框中选择“mydynp06”，单击“Finish”按钮，如图 4-253 所示。

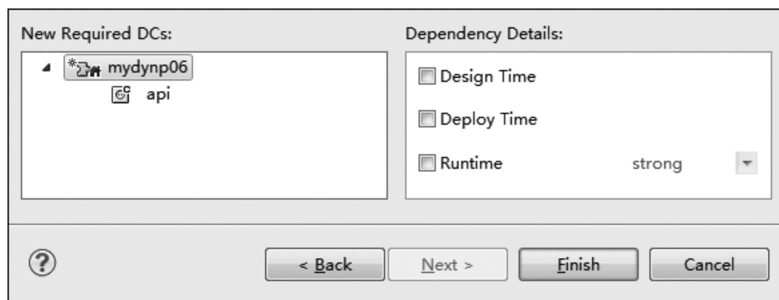


图 4-253　选择依赖的具体属性

10）依赖配置完成后，进入“mydynp05”项目，新建视图“Dynp05_01Comp_03View”，随后右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“ViewContainerUIElement”视图容器类型，输入 ID 名称，单击“OK”按钮，如图 4-254 所示。

11）完成元素添加后，可以在视图中看到预览效果，如图 4-255 所示。



图 4-254　元素清单

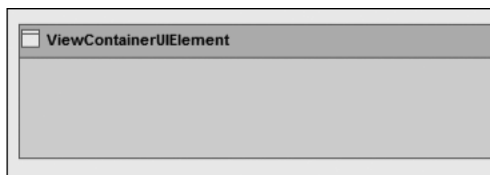


图 4-255　预览效果

12）双击打开“mydynp05”中对应的窗口，就能看到嵌入的容器，如图 4-256 所示。

13）右击容器，选择“Embed Interface View Of a Component Instance”嵌入一个组件实例的接口视图，在弹出的对话框中添加对应组件，单击“Add”按钮，如图 4-257 所示。

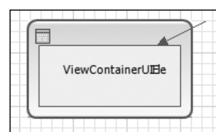


图 4-256　嵌入视图后的预览效果

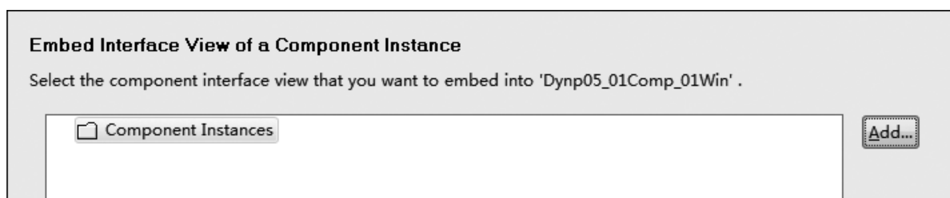


图 4-257　选择对应接口组件

14）在弹出的对话框中选择“Dynp06_01Comp”组件，随后单击“OK”按钮，如图 4-258所示。

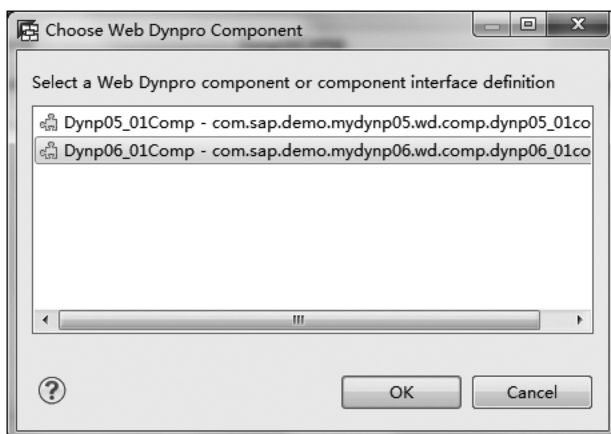


图 4-258　选择对应组件

15）输入对应名称与相关属性，随后单击“Finish”按钮，如图 4-259 所示。

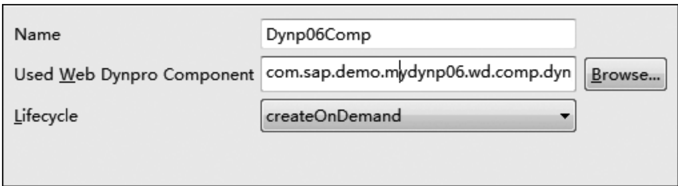


图 4-259　输入引用组件名称

16）选中新添加的接口视图，随后单击“Finish”按钮，如图 4-260 所示。

17）完成配置后，可以在模型编辑器中看到嵌入后的状态，如图 4-261 所示。



图 4-260　选择接口视图

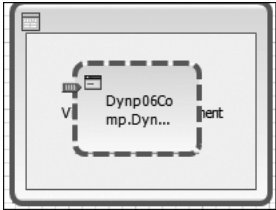


图 4-261　嵌入后预览效果

18）分别编译和部署“mydynp06”和“mydynp05”项目并运行，可以看到容器中显示了“mydynp05”中视图的内容，如图 4-262 所示。

审批历史

审批人	审批结果	审批意见	审批日期

图 4-262　运行效果

4.4　特殊元素

4.4.1　定时器

常用的页面定时触发器（TimedTrigger）在页面中是不可见的，触发的时间间隔根据延时参数（delay）来进行设置。

它包含以下属性。

- delay：触发器执行的延时间隔，以秒为基准单位，当延时设置为“0”时，定时触发器将会关闭。
- enabled：控制元素是否可用。
- onAction 事件：用于绑定定时触发的动作。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择关键字下拉列表类型，

输入 ID 名称，单击“OK”按钮，如图 4-263 所示。

2）建立上下文特性“delay”，类型为整型，如图 4-264 所示。

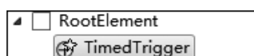


图 4-263　元素清单

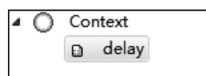


图 4-264　上下文清单

3）绑定上下文特性至定时器的延时属性中，如图 4-265 所示。

4）创建定时器触发的动作“Triggered”，如图 4-266 所示。

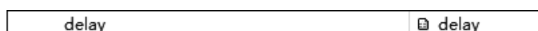


图 4-265　修改属性



图 4-266　修改事件属性

5）编写动作逻辑，代码如下：

```
wdComponentAPI. getMessageManager ( ) . reportSuccess ( " TimeTriggered!" + System. currentTimeMillis( ) );
```

6）编写初始化代码，设定 delay 值：

```
wdContext. currentContextElement( ). setDelay(2);          //初始化时设定延时时间为 2 秒
```

7）保存所有修改后，编译并部署项目，运行后就能看到测试效果。页面每隔两秒会打印当前时间，如图 4-267 所示。

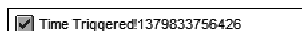


图 4-267　运行效果

4.4.2　图片

图片包含以下属性。

- adjustImageSize：控制图像尺寸是否可调整。
- border：控制边框大小。
- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- enabled：控制元素是否可用。
- height：指定图像高度。
- isDecorative：控制图片是否进行修饰。
- isDragHandle：控制是否可以进行拖曳。
- source：控制图片的路径。
- visible：控制元素是否可见。
- width：指定图像宽度。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择关键字下拉列表类型，

输入 ID 名称，随后单击“OK”按钮，如图 4-268 所示。

2）建立上下文特性，类型为字符串，如图 4-269 所示。

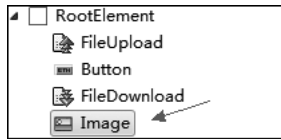


图 4-268　元素清单

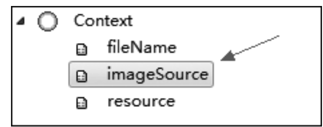


图 4-269　上下文清单

3）绑定上下文结点至元素的“source”属性中，如图 4-270 所示。

4）完成所有修改后，预览视图，如图 4-271 所示。

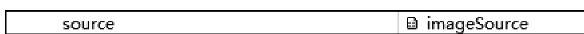


图 4-270　修改属性

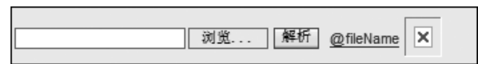


图 4-271　运行效果

5）解析时添加处理资源的代码：

```
String url = resource. getUrl( WDFileDownloadBehaviour. OPEN_INPLACE. ordinal( ) ) ;//图片显示
wdContext. currentContextElement( ). setImageSource( url) ;
```

6）保存所有修改，并重新编译、运行之后，即可进行相关测试。单击“浏览”按钮，选择图片后，单击“解析”按钮，如图 4-272 所示。

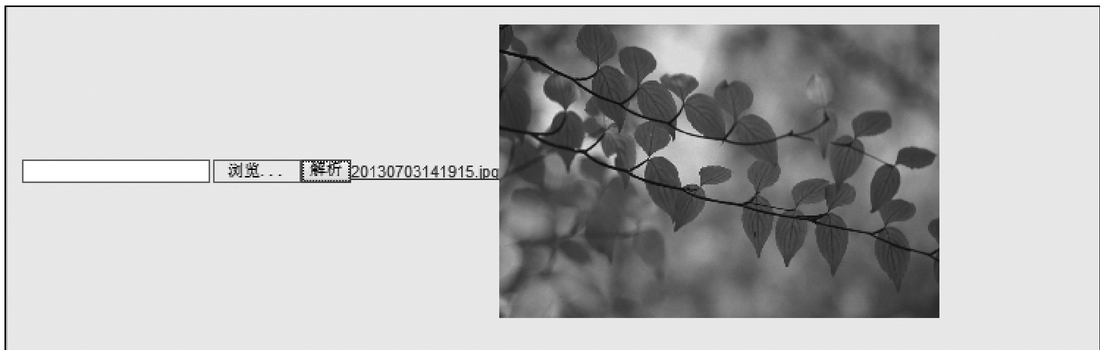


图 4-272　运行效果

注：本例在 FileUpload 控件实例之上，图片支持 GIF、JPG、PNG 等格式。

7）解析逻辑的全部代码如下：

```
IWDResource resource = wdContext. currentContextElement( ). getResource( ) ;
if( resource! = null) {
String resourceName = resource. getResourceName( ) ;
wdComponentAPI. getMessageManager( ). reportSuccess(
"打印 ResourceName:" + resourceName) ;
wdContext. currentContextElement( ). setFileName( resourceName) ;
//以下为解析 TxT 文件
try{
```

```
InputStream read = resource. read( true) ;                    //resource 读取文件输入流
InputStreamReader reader = new InputStreamReader(
read, "GBK") ;                                                //根据文件输入流构造 Reader
BufferedReader bufferedReader = new BufferedReader( reader) ;
String lineTxt = null;
while( ( lineTxt = bufferedReader. readLine( ) ) ! = null) {  //逐行解析文件
//wdComponentAPI. getMessageManager( ). reportSuccess(
//"文件内容:" + lineTxt) ;                                    //打印文件内容
}
//图片显示
String url = resource. getUrl( WDFileDownloadBehaviour. OPEN_INPLACE
. ordinal( ) ) ;
wdContext. currentContextElement( ). setImageSource( url) ;
read. close( ) ;                                              //关闭文件输入流
} catch( IOException e) {
//TODO Auto - generated catch block
//e. printStackTrace( ) ;
wdComponentAPI. getMessageManager( ). reportException( "解析的文件异常!" ) ; //解析文件异常
}
} else{
wdComponentAPI. getMessageManager( ). reportException( "解析的文件为空!" ) ; //未浏览选择文件
}
```

4.4.3 网页框架

IFrame 是嵌在页面内的框架元素，可以显示 HTML 等 Web 资源，通过设置绑定的 URL，控件能够自动进行内容切换。

它包含下列属性。

- height：设置高度。
- lifeCycle：设置生命周期。
- postParameters：当 postParameters 有值时，请求以 POST 方式传递。
- scrollingMode：设置 IFrame 是否带滚动条。
- source：控制 URL 地址。
- supportsResizing：是否支持自动调整大小。
- width：设置宽度。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“IFrame”框架类型，输入 ID 名称，随后单击“OK”按钮，如图 4-273 所示。

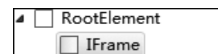


图 4-273 元素清单

2）建立上下文特性“url”，类型为字符串类型，如图 4-274 所示。

3）随后绑定该特性到元素的“source”属性中，如图 4-275 所示。



图 4-274 上下文清单

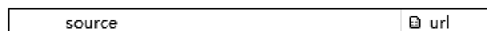


图 4-275 修改属性

4）初始化赋值 url：

```
//获取 request 对象
IWDRequest request = WDProtocolAdapter. getProtocolAdapter( )
. getRequestObject( ) ;
//初始化设置 url
String url = "http://" + request. getServerName( ) + ":"
 + request. getServerPort( ) + "/" ;
wdContext. currentContextElement( ). setUrl( url) ;
```

5）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-276 所示。

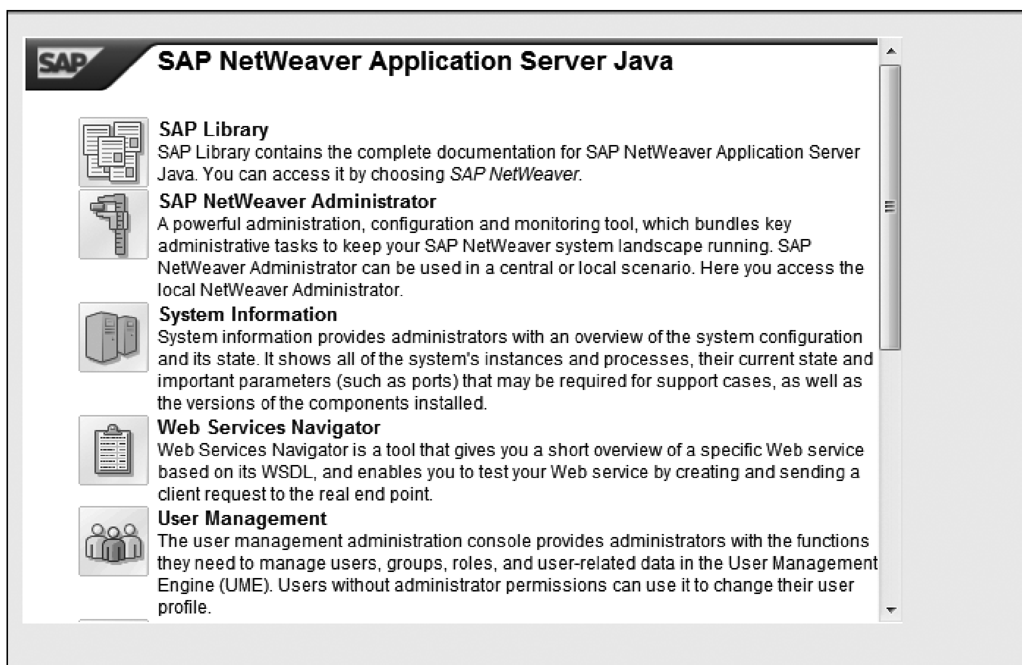


图 4-276　运行效果

注：在 Web Dynpro 中 IFrame 已被标记为过时的元素，除特殊情况下不建议使用。

4.4.4　Web 部件

在 NetWeaver 7.3 版本之后增加了 Web 部件（WebWidget）元素，该元素可以嵌入自定义 HTML 内容，并且可以添加脚本开发（JavaScript，JS）。对于 Web Dynpro 中标准元素无法实现的功能，该元素实现较大的扩展与延伸。

Web 部件包含了下列属性。

- contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。
- contextMenuId：设置上下文菜单的唯一标识。
- enabled：控制元素是否可用。
- html：自定义 HTML 内容。
- scriptOnLoad：元素加载或产生请求时执行脚本。

- scriptOnUnLoad：页面关闭或元素销毁时执行的脚本。
- style：页面样式。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“WebWidget”类型，输入 ID 名称，随后单击“OK”按钮，如图 4-277 所示。

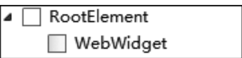


图 4-277　元素清单

2）建立上下文特性，分别用于存放网页内容、加载脚本和显示样式，如图 4-278 所示。

3）绑定属性至元素，如图 4-279 所示。

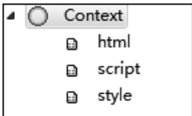


图 4-278　上下文清单

html	html
scriptOnLoad	script
scriptOnUnload	
style	style

图 4-279　修改属性

4）在视图的“wdDoInit”初始化方法中添加以下代码：

```
//赋值 html
wdContext. currentContextElement( ). setHtml(
" < button type =' button' id =' button' > Show Hello World < /button > ");
//赋值脚本
wdContext
. currentContextElement( )
. setScript( " var btn = UCF_DomUtil. $ ( ' button' ) ;"
 + " UCF_DomUtil. attachEvent( btn,' click' , function( ) { alert( ' Hello World' ) ; } ) ;" ) ;
//赋值 style
wdContext. currentContextElement( ). setStyle( " background - color: red' " ) ;
```

5）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-280 所示。



图 4-280　运行效果

4.4.5　路线图

路线图（RoadMap）将若干步骤显示在一个顺序的序列中，如可以在向导中进行具体步

骤的提醒，并能够根据不同需求设置样式。路线图包含了两种子元素："RoadMapStep" 步骤和 "MultipleRoadMapStep" 多级步骤，在有固定步数的路线图中可直接使用前者，而在动态的路线图（不确定步数）中则可使用后者。两种子元素也可以混合使用，非常灵活。路线图元素包含下列属性。

1）accessibilityDescription：设置复选框组中的文本是否在用户选中整个元素时显示。

2）contextMenuBehaviour：设置上下文菜单行为，默认有"继承""允许"和"禁止"3 种类型。

3）contextMenuId：设置上下文菜单的唯一标识。

4）enabled：控制元素是否可用。

5）endPointDesign：控制路线图终点的设计样式，默认有以下 5 种样式。

- disabled：选中路线图终点时不会触发任何事件，且显示为未激活状态。
- more：路线图终点表示还有更多的后续步骤未显示。
- moreDisabled：路线图终点表示还有更多的步骤，但是终点的样式是未激活状态。
- selected：路线图终点可以被选中，还能额外显示一些文本信息。
- standard：默认标准的路线图终点的样式。

6）selectedStep：控制默认选中的步骤，若使用多步骤路线图，则必须与相应的上下文结点进行绑定。

7）startPointDesign：控制路线图起点的设计样式，与终点样式类似，默认有以下 5 种样式。

- disabled：选中路线图起点时不会触发任何事件，且显示为未激活状态。
- more：路线图起点表示当前显示的流程之前还有更多的步骤未显示。
- moreDisabled：路线图起点表示之前还有更多的步骤，但是显示样式是未激活状态。
- selected：路线图起点可以被选中，还能额外显示一些文本信息。
- standard：默认标准的路线图起点的样式。

8）tooltip：控制提示消息的显示文本。

9）visible：控制元素是否可见。

10）onLoadSteps 事件：当路线图还有更多步骤未显示时，用户在单击更多步骤时进行的初始化事件，可以通过映射判断是否已到达路线图的末尾。

11）onSelect 事件：会在用户单击任一步骤时触发，可以通过映射返回当前选中步骤的唯一标识。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击 "Outline" 元素清单中的 "RootElement" 根结点，选择插入子对象，选择 "RoadMap "路线图类型，输入 ID 名称，随后单击 "OK" 按钮，如图 4-281 所示。

2）接着插入路线图中的步骤，右击 "Outline" 中新建的路线图，选择 "Insert Step"，随后选择 "MultipleRoadMapStep" 多级步骤，如图 4-282 所示。

3）完成创建后，保存所有修改，如图 4-283 所示。

4）创建一个上下文结点以及字符串类型的特性，分别用于对应路线图的数据源和存放步骤的文本，如图 4-284 所示。

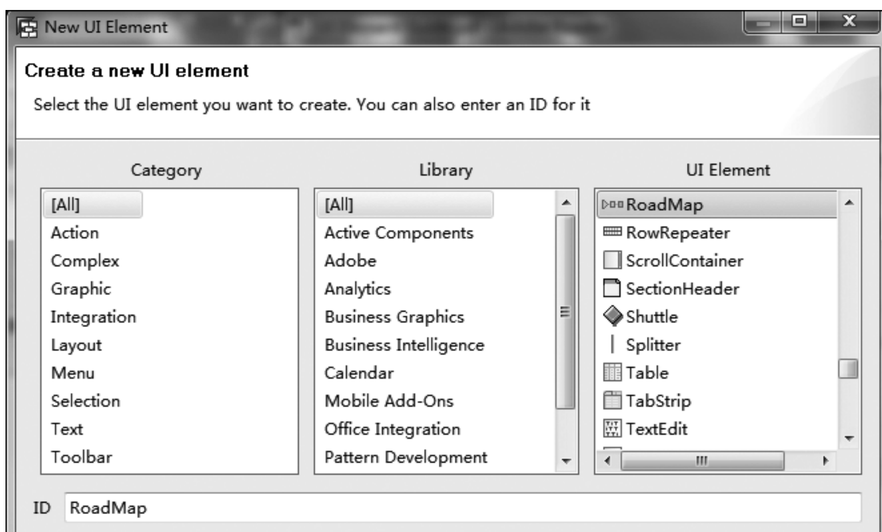


图 4-281　选择元素类型

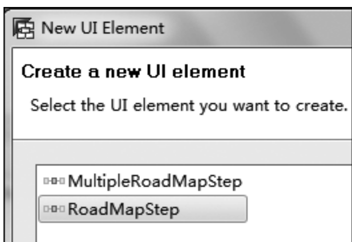


图 4-282　选择元素类型

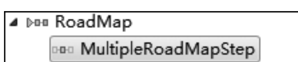


图 4-283　元素清单



图 4-284　上下文清单

5）由于实例中进行测试的是多级步骤，所以结点的集合属性必须是“0··n”或“1··n”，如图 4-285 所示。

6）随后选中“MultipleRoadMapStep”，修改其属性。将“dataSource”选择刚创建的“roadMap”结点，将“description”与“desc”特性进行绑定。注意不要混淆其中的两个属性，“description”属性是为每一步骤显示的描述，而“name”则是为每一步骤显示的名称，这里只绑定描述属性，如图 4-286 所示。

Misc	
Collection Cardinality	0..n
Initialize Lead Selection	true

图 4-285　修改结点属性

dataSource	roadMap
description	roadMap.desc

图 4-286　修改属性

7）在开发工具中，可以看到整个路线图的雏形预览，如图 4-287 所示。

8）随后选择“RoadMap”路线图，修改其属性。将“selectedStep”选中步骤修改为“MultipleRoadMapStep”。保存修改后，路线图中就会有默认选中并且高亮显示的步骤，如图 4-288所示。



图 4-287　预览效果



图 4-288　预览效果

9）为路线图创建一个“Select”动作并与“onSelect”事件绑定，如图 4-289 所示。

◢ Events	
onLoadSteps	
onSelect	Select

图 4-289　修改事件属性

10）在创建动作时，新建“id”和“ele”两个参数，用于存放对应步骤的唯一标识和步骤对象，如图 4-290 所示。

Details | Parameters

Name	Type
id	string
ele	com.sap.tc.webdynpro.progmodel.api.IWDNodeElement

图 4-290　添加参数

11）右击“Outline”清单中的路线图元素，选择“Parameter Mapping”创建参数映射（如果在创建动作时不添加对应参数，则无法进行映射），将路线图所选的关步骤 ID 与创建的动作中的参数“id”进行映射，将所选对象与参数“ele”进行映射，随后单击“OK”按钮，如图 4-291 所示。

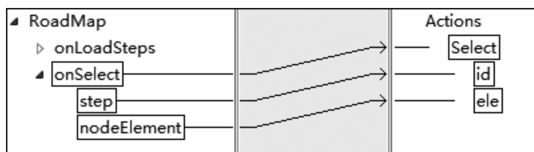


图 4-291　进行参数映射

12）保存所有修改后进入 Java 编辑器，添加事件处理器的逻辑，打印当前选中步骤的标识：

```
wdComponentAPI. getMessageManager( ). reportSuccess( "Selected Step:" + id);
wdContext. nodeRoadMap( ). setLeadSelection( ele. index( ) );        //设置为选中步骤
```

13）在“wdDoInit”方法中添加初始化路线图的代码：

```
IRoadMapNode nodeRoadMap = wdContext. nodeRoadMap( ) ;
IRoadMapElement  createAndAddRoadMapElement  =  nodeRoadMap. createAndAddRoadMapElement
( ) ;
createAndAddRoadMapElement. setDesc( "步骤一" ) ;
createAndAddRoadMapElement = nodeRoadMap. createAndAddRoadMapElement( ) ;
createAndAddRoadMapElement. setDesc( "步骤二" ) ;
createAndAddRoadMapElement = nodeRoadMap. createAndAddRoadMapElement( ) ;
createAndAddRoadMapElement. setDesc( "步骤三" ) ;
```

14）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-292 所示。



图 4-292　运行效果 1

15）单击任意步骤，如图 4-293 所示。

16）在消息区域中即可提示对应选中的步骤标识，如图 4-294 所示。

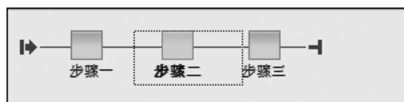


图 4-293　运行效果 2

Selected Step:MultipleRoadMapStep

图 4-294　选择后的效果

4.4.6 阶段指示器

阶段指示器（PhaseIndicator）与路线图元素类似，也是将若干步骤显示在一个顺序的序列中，只是每一个步骤在这里被替换成了一个阶段（phase），它的子元素也有“Phase”阶段和“MultiplePhase”多级阶段两种类型。不同于路线图元素的是，每个阶段可以有状态图标的提醒（完成、警告以及不可用），而且阶段指示器更多地用于显示更“大”的步骤。一般来说，完成一个阶段会比完成一个步骤花费更多的时间，这是阶段指示器的定位。它包含了下列属性。

- accessibilityDescription：设置消息提示和描述等文本是否在用户选中整个选项卡集时显示。
- backgroundDesign：控制阶段指示器的背景样式，默认有“emphasized”和“transparent”两种样式。
- contextMenuId：设置上下文菜单的唯一标识。
- controlMenuDisabling：控制菜单是否可用。
- enabled：控制元素是否可用。
- firstVisiblePhase：用于指定可见的第一阶段。
- selectedPhase：控制器当前选中阶段的唯一标识。
- tooltip：控制提示消息的显示文本。
- visible：控制元素是否可见。
- onSelect 事件：选择事件，在用户单击任一阶段时触发。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“PhaseIndicator”阶段指示器类型，输入 ID 名称，随后单击“OK”按钮，如图 4-295 所示。

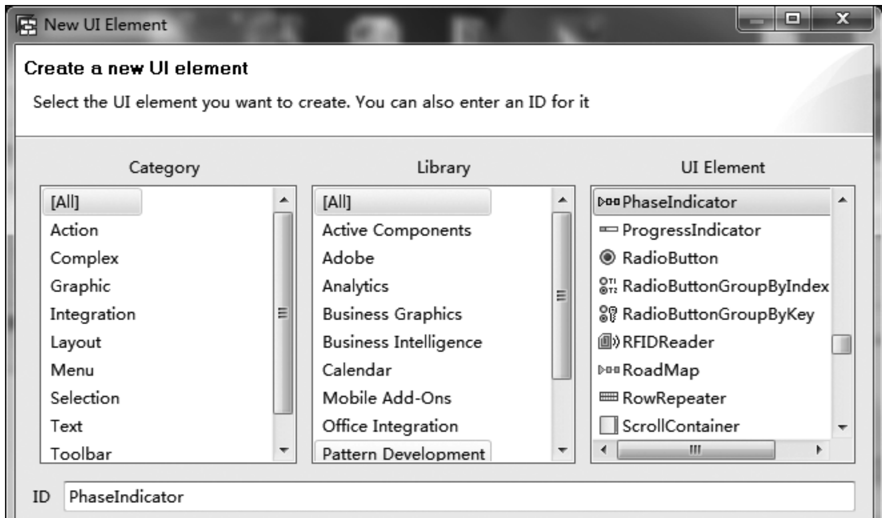


图 4-295 选择元素类型

2）随后在阶段指示器中添加子元素，如图 4-296 所示。

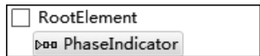


图 4-296 元素清单

3）接着插入阶段指示器图中的步骤，右击“Outline”中新建的路线图，选择“Insert Phase”，随后选择“MultiplePhase”多级

阶段，如图 4–297 所示。

4）完成创建后，保存所有修改，如图 4–298 所示。

5）创建一个上下文结点以及字符串类型的特性，分别用于对应阶段指示器的数据源和存放阶段的描述，如图 4–299 所示。

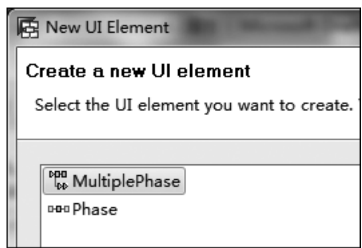


图 4–297　选择元素类型

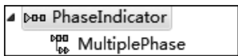


图 4–298　添加后的效果



图 4–299　上下文清单

6）由于本实例中进行测试的是多级阶段，所以结点的集合属性必须是“0··n”或“1··n”，如图 4–300 所示。

7）随后选中“MultiplePhase”多级阶段（不是阶段指示器），修改其属性。将“dataSource”选择刚创建的“phaseIndicator”结点，将“description”与“desc”特性进行绑定。注意，不要混淆其中的两个属性，“description”属性是为每一步骤显示的描述，而“name”则是为每一步骤显示的名称，这里只绑定描述属性，如图 4–301 所示。

Misc	
Collection Cardinality	0..n
Initialize Lead Selection	true

图 4–300　结点属性

dataSource	phaseIndicator
description	phaseIndicator.desc

图 4–301　修改属性

8）随后选择“PhaseIndicator”阶段指示器，修改其属性。将“selectedPhase”选中阶段修改为“MultiplePhase”。保存修改后，阶段指示器中就会有默认选中并且高亮显示的阶段，如图 4–302 所示。

selectedPhase	MultiplePhase

图 4–302　修改属性

9）在“wdDoInit”方法中添加初始化路线图的代码：

```
IPhaseIndicatorNode nodePhaseIndicator = wdContext. nodePhaseIndicator( ) ;
IPhaseIndicatorElement createAndAddPhaseIndicatorElement = nodePhaseIndica-
tor. createAndAddPhaseIndicatorElement( ) ;
createAndAddPhaseIndicatorElement. setDesc( "步骤一" ) ;
createAndAddPhaseIndicatorElement = nodePhaseIndicator. createAndAddPhaseIndicatorElement( ) ;
createAndAddPhaseIndicatorElement. setDesc( "步骤二" ) ;
createAndAddPhaseIndicatorElement = nodePhaseIndicator. createAndAddPhaseIndicatorElement( ) ;
createAndAddPhaseIndicatorElement. setDesc( "步骤三" ) ;
```

10）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4–303 所示。

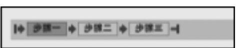


图 4–303　运行效果

4.4.7　进度指示器

进度指示器（ProgressIndicator）通过一个横向的进度条来显示一个动作的实际完成比

例。进度指示器可以用来标识一个问卷调查的提交比例，也可以用来标识一个长期项目的实际开发进度。它包含了下列属性。

1）barColor：控制进度指示器的填充颜色。

- critical：进度条默认显示为橙色。
- negative：进度条默认显示为红色。
- neutral：进度条默认显示为蓝色。
- positive：进度条默认显示为绿色。

注：以上颜色都是在默认主题下的颜色，实际项目中颜色也会根据不同的 SAP 门户主题发生变化。

2）contextMenuBehaviour：设置上下文菜单行为，默认有“继承”“允许”和“禁止”3 种类型。

3）contextMenuId：设置上下文菜单的唯一标识。

4）displayValue：控制进度指示器上显示的文本，默认会显示进度条的百分比数字。

5）enabled：控制元素是否可用。

6）percentValue：控制进度指示器的百分比数值。

7）showValue：控制是否显示“displayValue”属性中设置的文本。

8）tooltip：控制提示消息的显示文本。

9）visible：控制元素是否可见。

10）width：控制元素的显示宽度。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择“ProgressIndicator”进度指示器类型，输入 ID 名称，随后单击“OK”按钮，如图 4-304 所示。

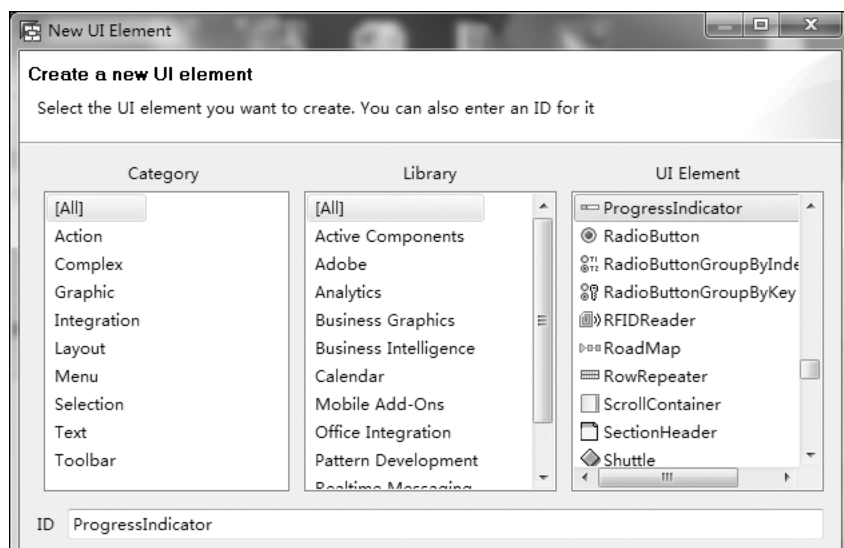


图 4-304　选择元素类型

2）随后创建视图的上下文，右击“Context”上下文，在弹出的对话框中输入特性名称并选择对应类型，这里输入“percentValue”并选择整数型，如图 4-305 所示。

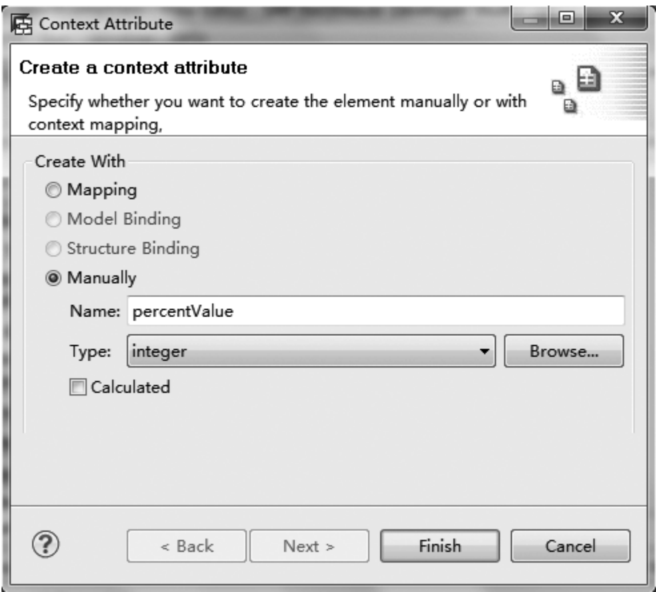


图 4-305　输入特性信息

3）单击“Finish”按钮，完成上下文特性的创建，如图 4-306 所示。

4）选择视图“Outline”中的进度控制器，将它的百分比属性与上下文特性进行绑定，如图 4-307 所示。注意，如果特性不是整数型，则将无法绑定。

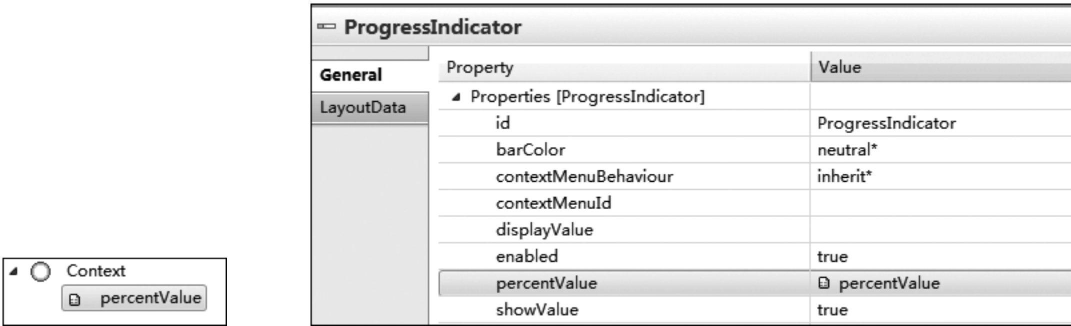


图 4-306　上下文清单　　　　图 4-307　修改属性

5）随后在进度指示器前面添加一个按钮，并创建默认动作，用于模拟进度变化。进入 Java 编辑器，在动作的事件处理方法中添加下列代码：

```
//使用上下文根结点的当前实例,并获取特性值
IContextElement currentEle = wdContext. currentContextElement( ) ;
int percentValue = currentEle. getPercentValue( ) ;
//通过特性值进行判断,进度条需要重置或增加
if( percentValue = = 100 ) {
    currentEle. setPercentValue( 0 ) ;
} else{
```

```
        currentEle. setPercentValue( percentValue + 10) ;
    }
```

6）保存所有修改，并重新编译、运行之后就可以进行测试了，如图 4-308 所示。

7）单击“运行”按钮，就能看到进度条与数值的变化，如图 4-309 所示。

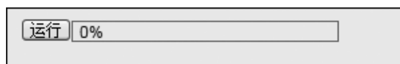


图 4-308　运行效果

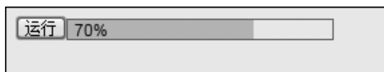


图 4-309　单击“运行”按钮的效果

4.4.8　菜单路径

菜单路径（BreadCrumb）常用于页面浏览历史。

菜单路径包含以下属性。

- behaviour：链接方式，包含 multipleLinks 和 singleLink 选项。
- enabled：控制元素是否可用。
- separatorImageSource：图片分隔符。
- separatorText：文本分隔符。
- visible：控制元素是否可见。
- onSelect 事件：用户选择某一页签时触发。

开发实例：

1）打开任一视图（如果需要创建新的工程或视图，请参考之前章节步骤），右击“Outline”元素清单中的“RootElement”根结点，选择插入子对象，选择关键字下拉列表类型，输入 ID 名称，随后单击“OK”按钮，如图 4-310 所示。

2）接着插入菜单路径中的步骤，右击“Outline”中新建的菜单路径，选择“Insert Step”，随后选择“MultipleBreadCrumbStep”多级步骤，如图 4-311 所示。

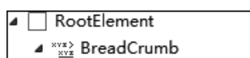


图 4-310　元素清单

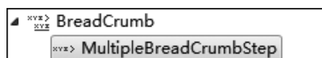


图 4-311　元素清单

3）建立上下文结点和特性，如图 4-312 所示。

4）绑定上下文结点至元素的数据源，如图 4-313 所示。

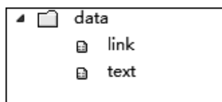


图 4-312　上下文清单

Properties[MultipleBreadCrumbStep]	
id	MultipleBreadCrumbStep
dataSource	data
text	data.text

图 4-313　修改属性

5）编写初始化代码：

```
IDataNode nodeData = wdContext. nodeData( ) ;
IDataElement createAndAddDataElement = nodeData. createAndAddDataElement( ) ;
createAndAddDataElement. setText( "导航一" ) ;
createAndAddDataElement. setLink( "url1" ) ;
```

```
createAndAddDataElement = nodeData. createAndAddDataElement( ) ;
createAndAddDataElement. setText( "导航二" ) ;
createAndAddDataElement. setLink( "url2" ) ;
createAndAddDataElement = nodeData. createAndAddDataElement( ) ;
createAndAddDataElement. setText( "导航三" ) ;
createAndAddDataElement. setLink( "url3" ) ;
```

6）保存所有修改，并重新编译、运行之后，即可进行相关测试，如图 4-314 所示。

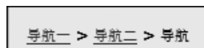


图 4-314　运行效果

第5章　综合场景应用

本章将根据 SAP 实施项目中的综合场景构造一个复杂应用，并模拟实际项目中构建应用的步骤，帮助大家进一步地了解 Web Dynpro 开发。

5.1　前期准备

在一个大型的实施项目中，前期准备十分重要，经验丰富的开发顾问也会参与其中的部分工作。

5.1.1　用户需求分析

虽然在一个 SAP 咨询项目中，需求梳理和确认主要由业务顾问与客户沟通来完成，但在确认之前还是需要开发顾问参与，因为开发人员可以对需求的技术可行性进行确认，也可以对某些复杂的场景设计进行必要的优化。这不仅有利于项目实施阶段顺利开展，也会帮助用户在上线后更快速地将程序应用到实际工作当中去。

在完成需求确认后，项目组成员需要对确认的需求进行分析，梳理业务逻辑、关联业务表和涉及的接口服务，最终形成需求规格说明书或功能设计说明书（根据实际项目安排决定），随后开发顾问就可以根据确认的需求进行实际的开发工作了。

本章开发实例的模拟场景为“采购申请查询”：用户根据若干查询条件搜索符合条件的采购申请，随后单击行项目也可查看采购申请行项目的明细。

5.1.2　接口服务设计

接口服务的方式有多种，在之前模型的章节中介绍过 RFC、EJB 以及 Web 服务等类型，当然对于某些外部数据源也可以直接访问数据库层 DAO 类并调用对应方法。不论使用哪种方式，Web Dynpro 中的模型最终都以接口的形式将各种接口服务转化为了 WD4J 可以识别的 Java 类。

本章将以 Web 服务（Web Service）作为接口，模拟 Web Dynpro 程序与后台数据的交互，包含采购申请查询以及采购申请行项目的明细查询。

开发实例：

1）首先创建一个测试用的 Web 服务，在 NWDS 中单击右上角的透视图切换按钮，切换到“Development Infratructure perspective”，如图 5-1 所示。

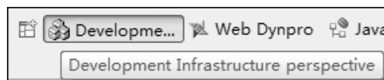


图 5-1　切换到 DI 透视图

2）右击自定义的 SC 目录“MyComponents”，选择创建一个“Development Component”，如图 5-2 所示。

3）在弹出的窗口中选择对应的 EJB 类型，单击“Next”按钮，如图 5-3 所示。

4）输入自定义 DC 的名称，单击“Next”按钮，如图 5-4 所示。

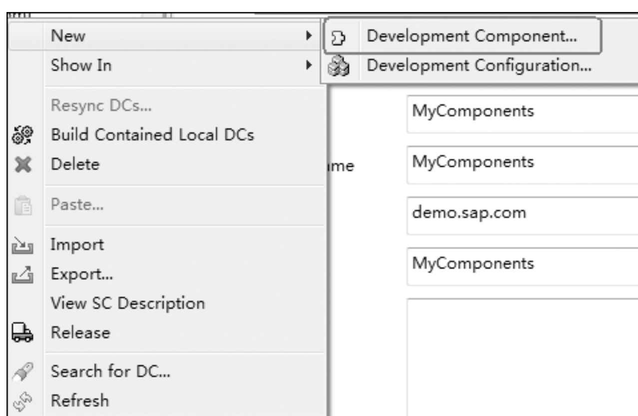


图 5-2　创建开发组件的菜单选择

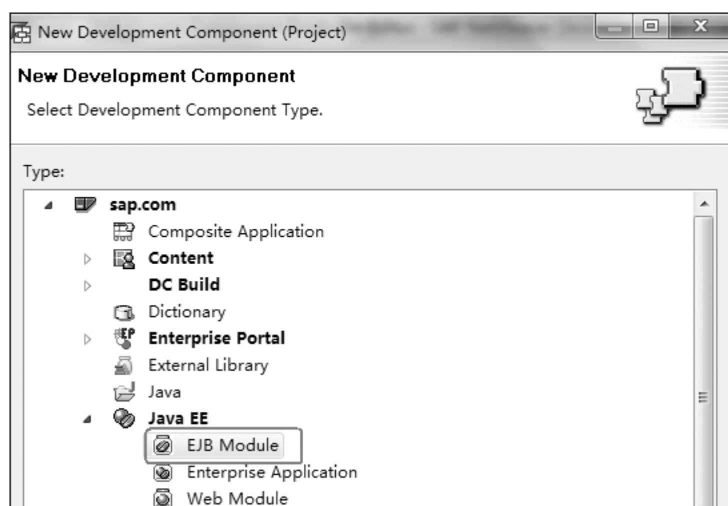


图 5-3　选择项目类型

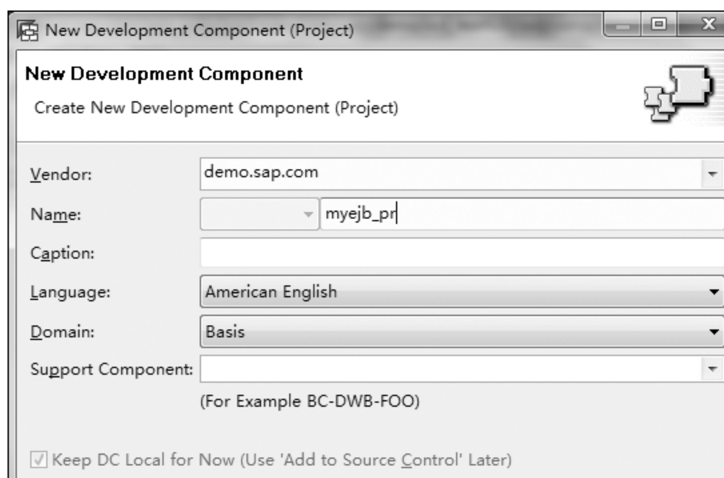


图 5-4　输入项目名称

5）选择对应的编译版本，如图 5-5 所示。

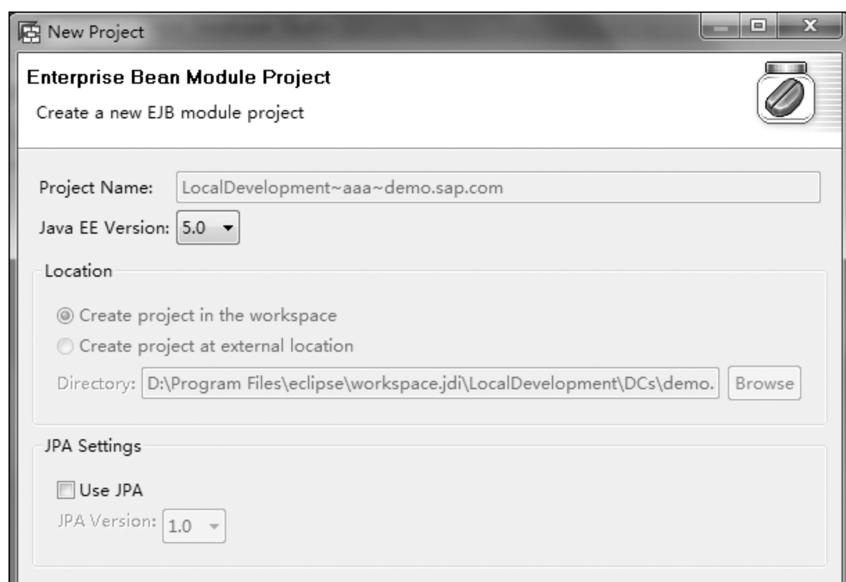


图 5-5　选择编译版本

6）创建完成后，进入 Java EE 透视图，并右击 EJB 工程，选择创建一个“Session Bean”，如图 5-6 所示。

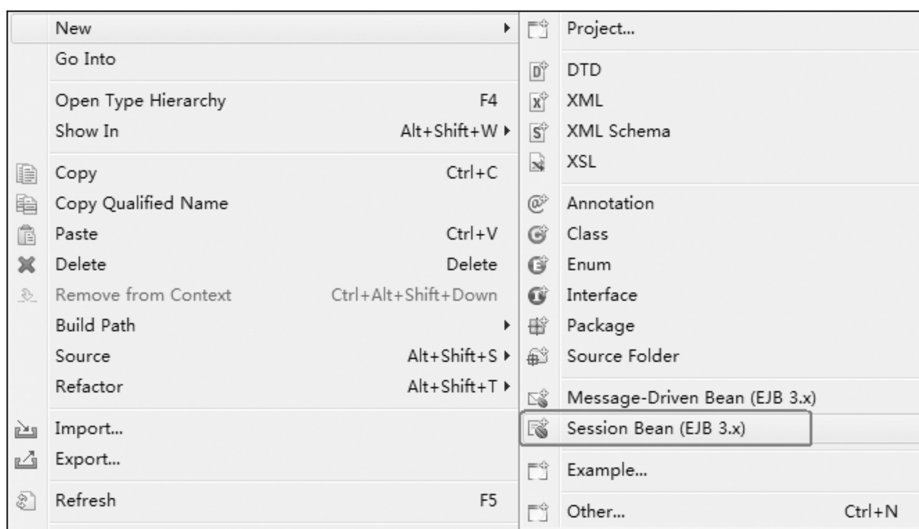


图 5-6　选择创建 SessionBean 菜单

7）按照命名规范输入自定义的包名和类名，单击“Finish”按钮，完成 EJB 创建，如图 5-7 所示。

8）创建完毕后，可以看到开发工具自动为 EJB 工程生成了一个空的类并同时实现了一个接口类。随后在“PurReqWs”类中添加一个方法，实例代码如图 5-8 所示。

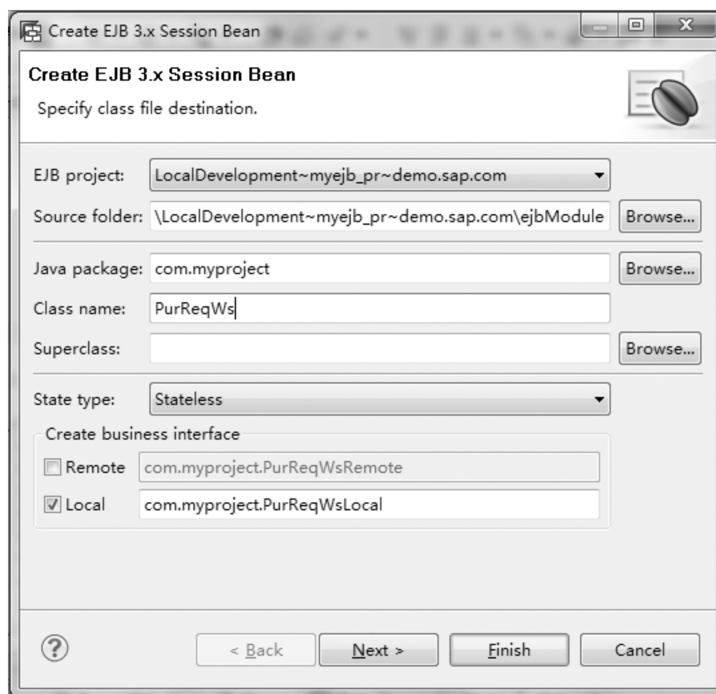


图 5-7　输入信息

```
public PurReq getPurReqs(String purReqNo, String purReqType){
    //TODO Auto-generated method stub
    return null;
}
```

```
public class PurReqWs implements PurReqWsLocal {

    /**
     * Default constructor.
     */
    public PurReqWs() {
        // TODO Auto-generated constructor stub
    }

    public PurReq getPurReqs(String purReqNo, String purReqType) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

图 5-8　添加代码

9）保存后在接口类中添加获取采购申请方法的声明（见图 5-9）：

```
public PurReq getPurReqs(String purReqNo, String purReqType);
```

```
@Local
public interface PurReqWsLocal {
    public PurReq getPurReqs(String purReqNo, String purReqType);
}
```

图 5-9　添加代码

10）由于还未定义采购申请类，因此暂时会有编译错误。下面需要根据需求（数据库定义）创建采购申请和采购申请条目的对象类。右击工程，选择创建一个程序包，如图 5-10所示。

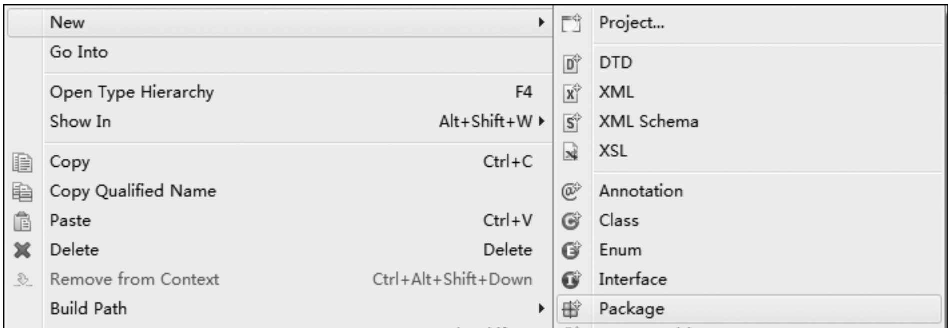


图 5-10　创建包的菜单选择

11）在随后弹出的对话框中输入需要创建的包名“com. bean”，并单击“Finish”按钮，如图 5-11 所示。

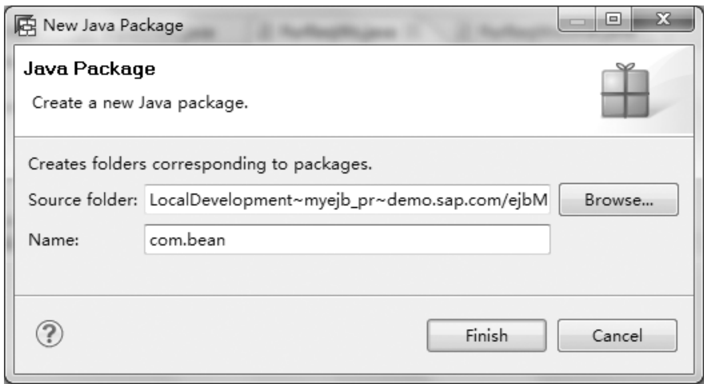


图 5-11　输入包名

12）右击“com. bean”包，选择创建一个类，如图 5-12 所示。

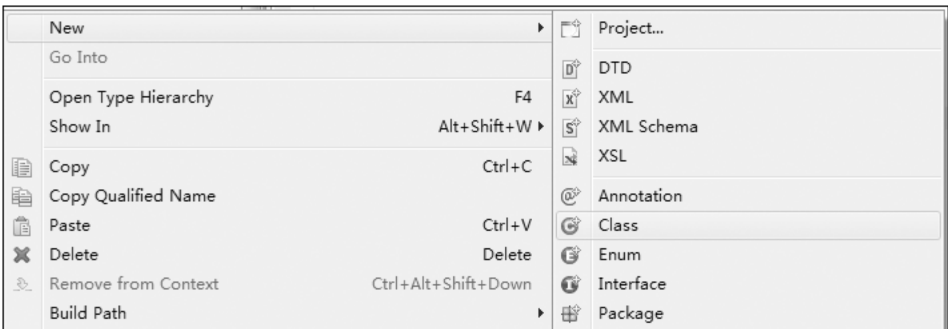


图 5-12　创建类的菜单选择

13）在弹出的对话框中输入需要创建的类名“PurReq”，并单击“Finish”按钮，如

图 5-13所示。

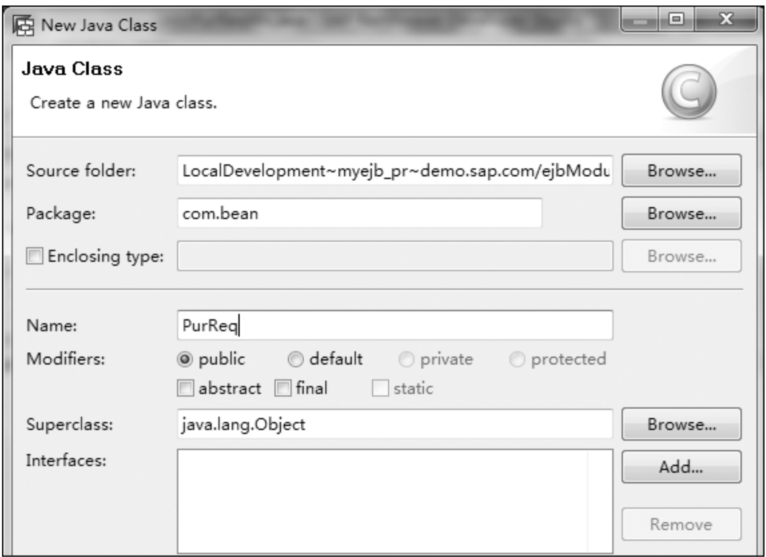


图 5-13　创建类

14）创建完成后，补充各个字段定义（暂时不需要添加 get 和 set 方法）：

```
public class PurReq{
    private String reqNo;                //申请单号
    private String purType;              //申请类型
    private String remark;               //备注
    private String desc;                 //描述
    private List<PurReqItem> items;      //行明细
}
```

15）定义字段后，在空白处右击，在弹出的快捷菜单中选择“Source”→“Generate Getters and Setters”，自动创建字段的 get 和 set 方法，如图 5-14 所示。

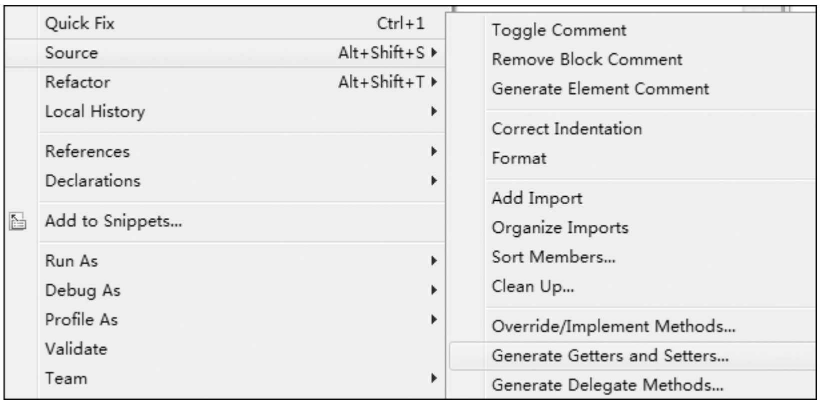


图 5-14　自动生成方法

16）在弹出的对话框中选中所有字段，随后单击“OK”按钮完成创建，如图 5-15 所示。

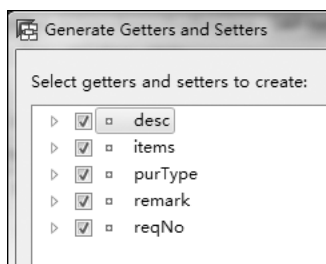


图 5-15　选择对应变量

17）此时暂时会有编译错误，因为还没有定义行项目的对象类。再创建一个类，命名为“PurReqItem”，并补充各个字段定义，并用同样的方法创建对应的 get 和 set 方法。字段定义如下：

```
public class PurReqItem{
private String reqNo;                    //申请单号
private String reqItemNo;                //项目编号
private String account;                  //科目
private String itemType;                 //项目类别
private String matCode;                  //物料
private String price;                    //单价
private String count;                    //数量
private String reqDate;                  //需求日期
private String unit;                     //单位
private String matGroup;                 //物料组
private String werks;                    //工厂
private String lgort;                    //库存
private String shortDesc;                //短文本
private String longDesc;                 //长文本
private String charg;                    //批次
private String deliveryDate;             //交货日期
private String lifnr;                    //供应商
private String ekorg;                    //采购组织
private String liwerks;                  //供应工厂
private String processStatus;            //处理状态
private String locks;                    //冻结标识
private String creator;                  //创建者
private String modifyDate;               //修改日期
private String reqPlanNo;                //需求计划号
private String adjustVendor;             //推荐供应商
private String isImp;                    //是否进口
private String currType;                 //资金类型
private String fileId;                   //技术文件 ID
private String deliveryAddr;             //交货地址
private String deliveryCity;             //交货城市
private String area;                     //国家/地区
}
```

18）创建完成后，保存所有修改即可，如图 5-16 所示。

19）下面基于当前 EJB 创建对应的 Web 服务。右击“PurReqWs”类，在菜单中选择“Web Services”→“Create Web service”，创建一个 Web 服务，如图 5-17 所示。

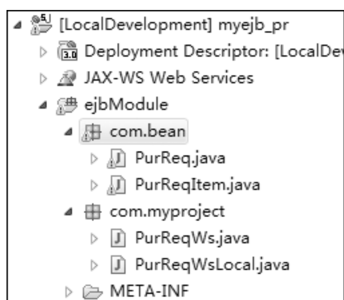


图 5-16　创建后项目清单

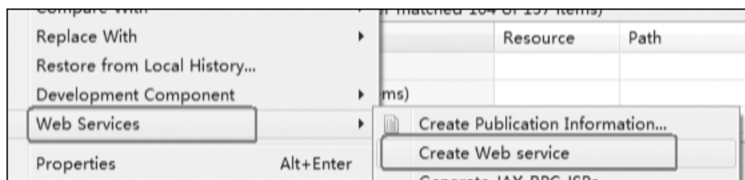


图 5-17　创建 Web 服务的菜单选择

20）在弹出的对话框中，将配置级别的滑块选择“Develop Service”，并单击“Next”按钮，之后的步骤均使用默认选项直至创建完毕，如图 5-18 所示。

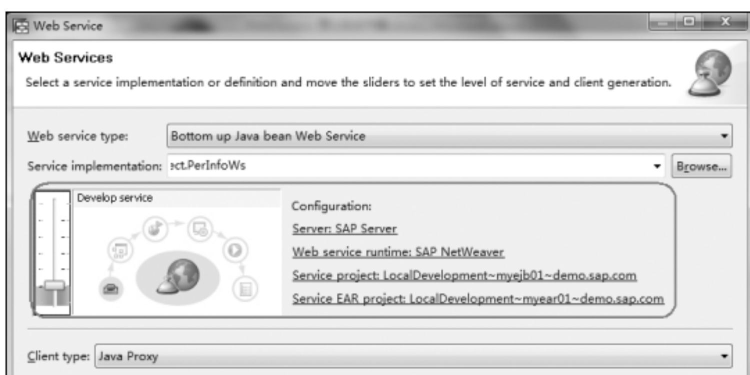


图 5-18　配置服务级别

21）再次进入 DI 透视图，创建一个 EAR 工程，用于依赖 EJB 工程并部署至服务器端。选择新建类型为“Enterprise Application”，并单击“Next”按钮，如图 5-19 所示。

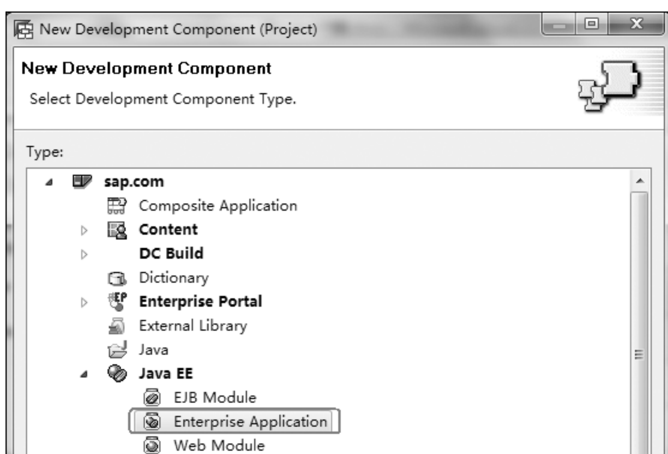


图 5-19　选择项目类型

22）根据实际的命名规范，输入自定义 ear 名称，并单击“Next”按钮，如图 5-20 所示。

23）根据实际需求，选择对应的 Java EE 版本，并单击“Next”按钮，如图 5-21 所示。

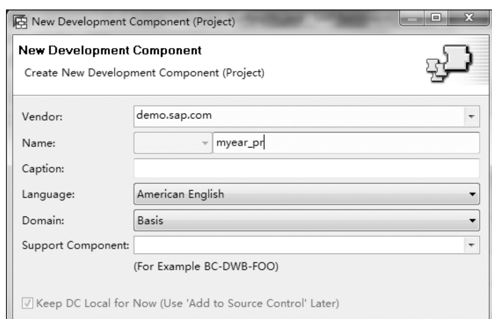


图 5-20　输入项目名称

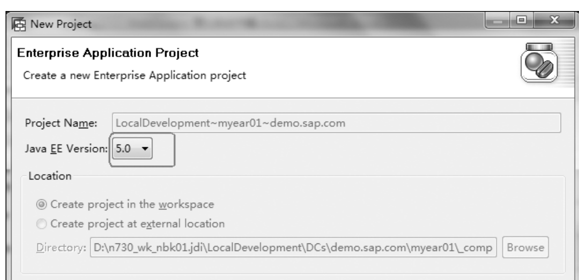


图 5-21　选择编译版本

24）选中需要进行依赖的 EJB 工程，单击“Finish”按钮，如图 5-22 所示。

25）保存所有修改，完成两个工程的创建，如图 5-23 所示。

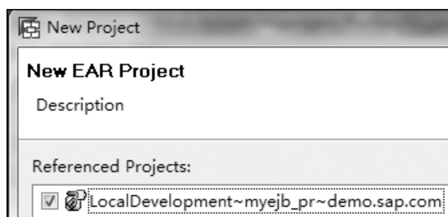


图 5-22　选择依赖工程



图 5-23　创建后项目目录

26）接着编译并部署 EAR 工程，将创建的 WebService 发布到服务器。右击“myear_pr”工程，选择“Development Component”→“Build”进行编译，编译完成后在同一菜单中选择“Deploy”进行部署，如图 5-24 所示。

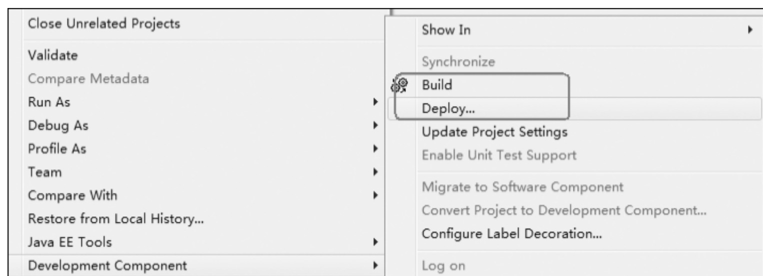


图 5-24　部署的菜单选择

27）验证 Web 服务是否部署成功。用管理员账号登录服务器 NWA 管理者平台，并进入“Configuration - Connectivity”→“Single Service Administration”，如图 5-25 所示。

28）在搜索条件中输入“*purreq*”并按〈Enter〉键，搜索创建的 Web 服务。在成功搜索到对应的服务后，找到下方的“WSDLs”页签，并单击“Test”按钮进行测试，如图 5-26 所示。

29）单击“Text”按钮会弹出一个新窗口，默认选中的就是定义的接口方法，单击“Next”按钮，如图 5-27 所示。

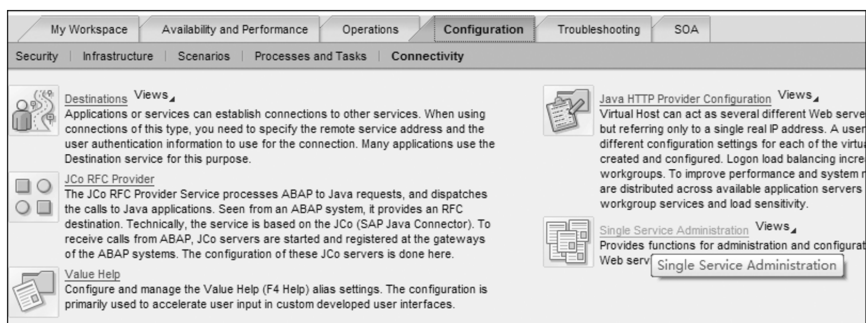


图 5-25　NWA 平台页面

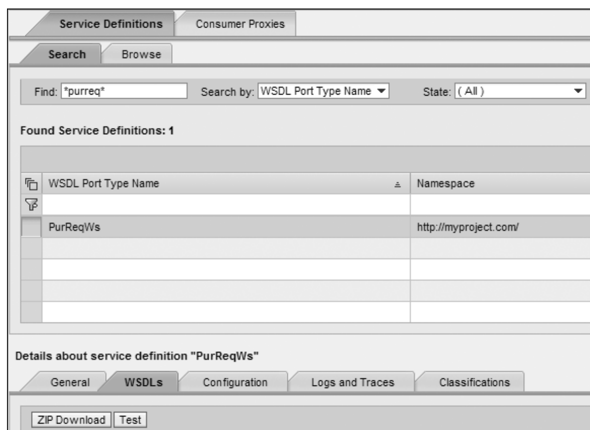


图 5-26　搜索对应的服务

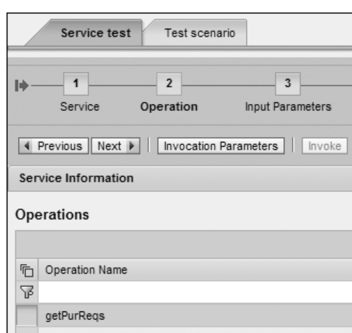


图 5-27　选择对应方法进行测试

30）在下一步中填写方法的输入参数，单击“Next”按钮，如图 5-28 所示。

31）得到了服务执行成功的消息提醒，暂时还没有返回值，如图 5-29 所示。

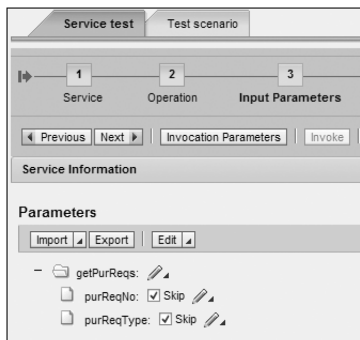


图 5-28　输入对应参数值

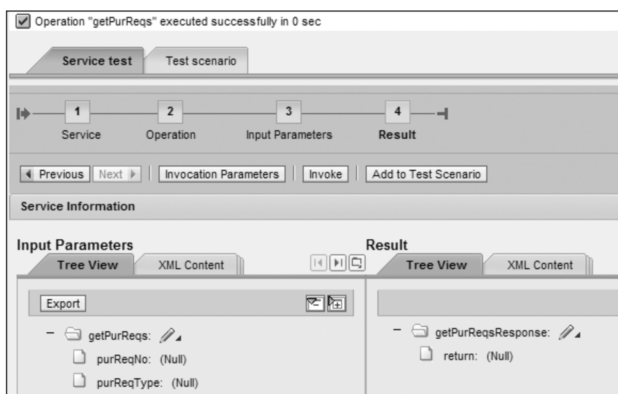


图 5-29　查看运行结果

32）回到 Web 服务中，修改之前的“getPurReqs”方法，导入必要的类文件。方法的实例代码如下：

```
public PurReq getPurReqs(String purReqNo, String purReqType) {
    PurReq req = new PurReq();
    req.setReqNo("010000010000");
    req.setPurType("01");
```

```
        req.setRemark("采购申请备注");
        req.setDesc("采购申请描述");
        List<PurReqItem> reqItems = new ArrayList<PurReqItem>();
        PurReqItem item = new PurReqItem();
        item.setAccount("1601030000");
        item.setAdjustVendor("10000000001");
        item.setArea("CN");
        item.setCharg("01");
        item.setCount("10");
        item.setCreator("00001001");
        item.setCurrType("01");
        item.setDeliveryAddr("西大街北四号");
        item.setDeliveryCity("南京");
        item.setDeliveryDate("20161111");
        item.setEkorg("0001");
        item.setFileId("001");
        item.setIsImp("1");
        item.setItemType("01");
        item.setLgort("Z001");
        item.setLifnr("0101000020");
        item.setLiwerks("0001");
        item.setLocks("0");
        item.setLongDesc("电缆\\00000065069548\\液压支架电液控制阀\\DBT\\进口");
        item.setMatCode("000000000010000315");
        item.setMatGroup("G001");
        item.setModifyDate("20120908");
        item.setPrice("2");
        item.setProcessStatus("A01");
        item.setReqDate("20120901");
        item.setReqItemNo("10");
        item.setReqNo("010000010000");
        item.setReqPlanNo("P0005");
        item.setShortDesc("电缆\\00000065069548\\液压支架电液控制阀\\DBT\\进口");
        item.setUnit("EA");
        item.setWerks("SD00");
        reqItems.add(item);
        req.setItems(reqItems);
        return req;
    }
```

33）再次编译并部署至服务器，进行 Web 服务的测试，即可得到硬编码的采购申请返回值，如图 5-30 所示。



图 5-30　运行效果

5.2　功能开发

根据需求分析可以明确 Web Dynpro 要实现表单查询以及行项目明细查询功能，而且这两个功能都涉及与接口服务的交互。

5.2.1　创建 WD4J 项目

本节将创建一个 Web Dynpro 工程，用于对 Web 服务的

调用。

开发实例：

1）在 Web Dynpro 透视图中，单击左上方新建按钮旁边的箭头，选择“Web Dynpro Development Component”，如图 5-31 所示。

2）在弹出的对话框中选择“MyComponents”，单击“Next”按钮，如图 5-32 所示。另外，也可以在 DI 透视图中右击“MyComponents”进行创建，再选择 Web Dynpro 类型，结果相同。

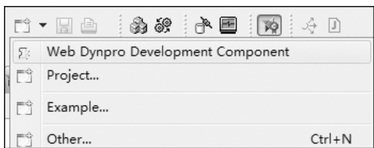


图 5-31　创建开发组件的菜单选择

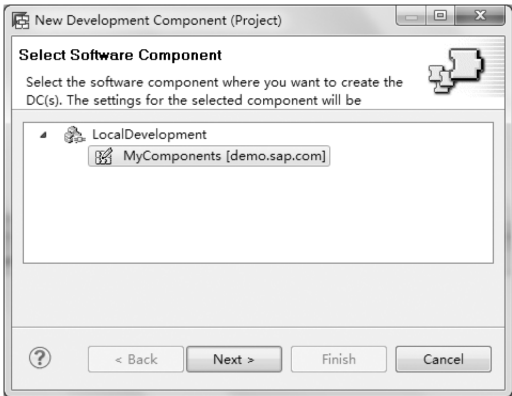


图 5-32　选择本地组件包

3）输入工程名称“mydynp04”，单击“Finish”按钮，完成创建，如图 5-33 所示。

4）随后展开工程的目录，右击“Components”组件，选择创建一个组件，如图 5-34 所示。

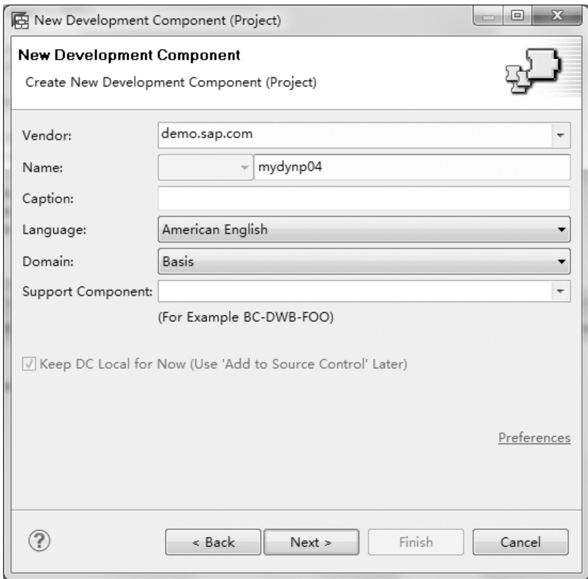


图 5-33　输入项目名称

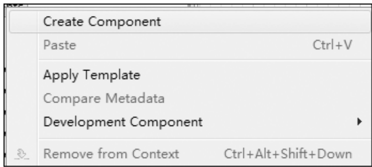


图 5-34　创建组件的菜单选择

5）输入组件名称，并勾选创建默认窗口和视图，单击“Finish”按钮，如图 5-35 所示。

6）至此，完成了一个 Web Dynpro 工程的创建，如图 5-36 所示。

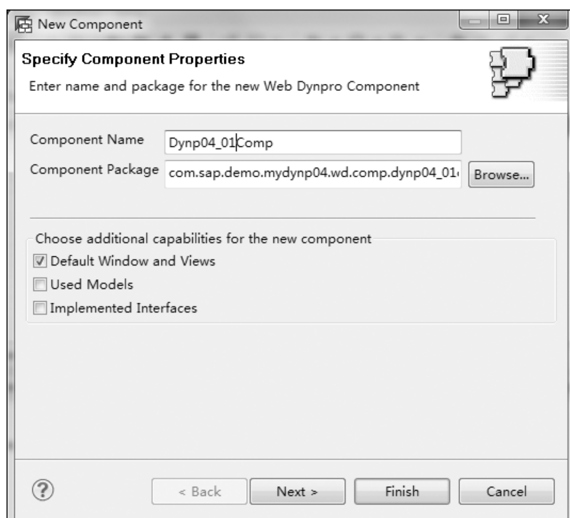


图 5-35 输入组件名称

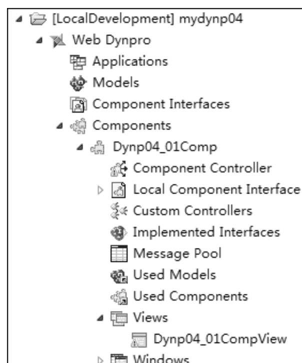


图 5-36 创建后的项目结构

5.2.2 核心功能开发

本节将调用之前定义的 Web 服务，并在 Web Dynpro 中完成处理。

开发实例：

1）首先创建对应模型，右击工程中的“Models”，选择“Create Model”，如图 5-37 所示。

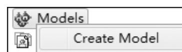


图 5-37 创建模型的菜单选择

2）在弹出的窗口中选择对应的模型类型，选择第二项“Adaptive Web Service Model”，并单击“Next”按钮，如图 5-38 所示。

3）输入自定义模型名称“NewAWS1Model”，并选中“Remote Location/File System”单选按钮作为 WSDL 描述文件的源文件，单击“Next”按钮，如图 5-39 所示。

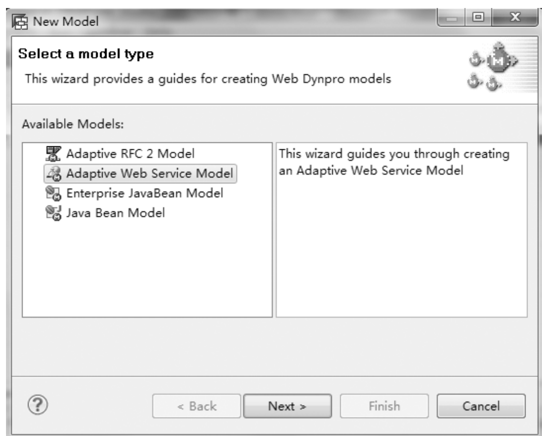


图 5-38 选择模型类型

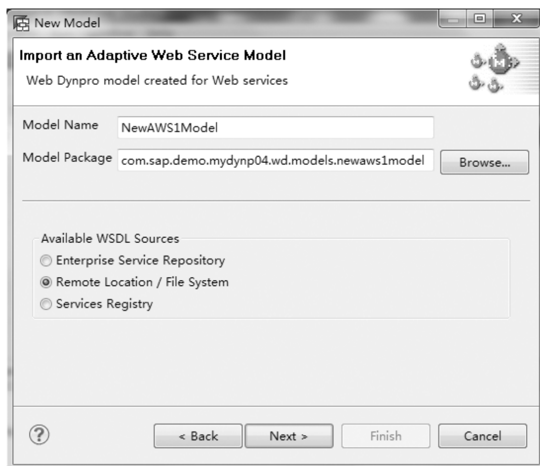


图 5-39 输入模型信息

4）输入之前创建的 Web 服务对应的 WSDL 地址（在 Service Definitions 中下方的“WS-DLS”页签中可以看到），粘贴的同时开发工具会立即对服务做连通性测试。测试成功后显示下一步，单击“Next”按钮，如图 5-40 所示。

5）接着进行运行时间相关配置，选中“No service group configuration”单选按钮，单击“Next”按钮，如图 5-41 所示。

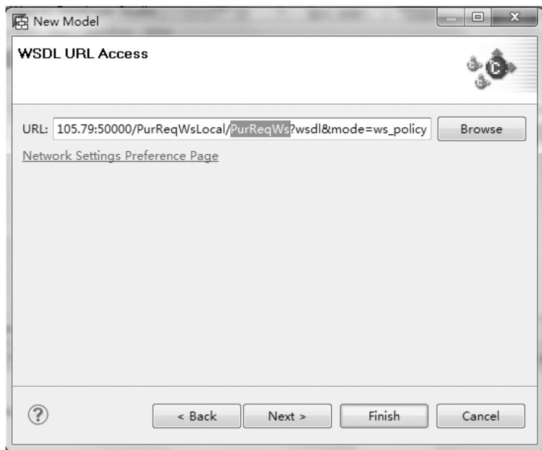


图 5-40　输入服务地址

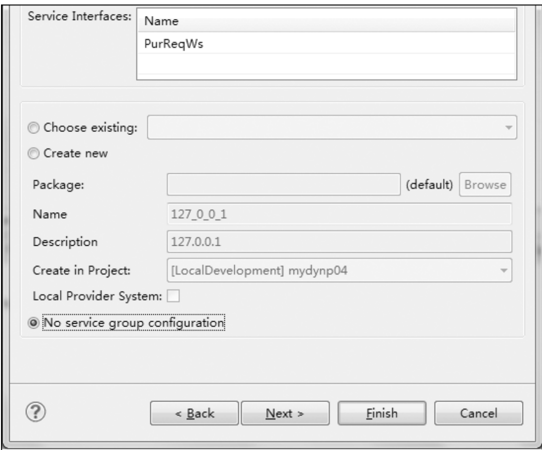


图 5-41　选择服务组

6）输入逻辑目标名称，使用默认配置，并单击“Next”按钮。如果已在服务器端配置完成，则可以使用所配置的名称，也可以使用同一个目的，如图 5-42 所示。

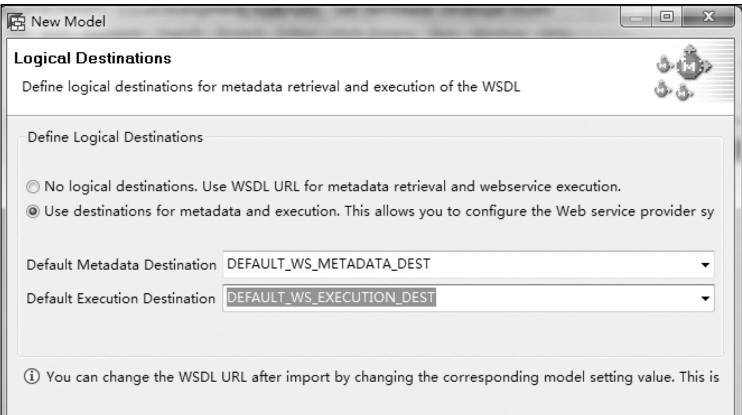


图 5-42　输入目的名称

7）根据需要对导入的模型类进行重命名，跳过则直接单击“Next”按钮，如图 5-43 所示。

8）在弹出的对话框中，可以看到导入日志，也就是看到对应 Web 服务接口在 Web Dynpro 模型中生成的接口结构，单击“Finish”按钮完成模型的导入配置，如图 5-44 所示。

9）完成配置后，就可以在工程的目录中看到对应导入的模型以及模型类结构，如图 5-45 所示。

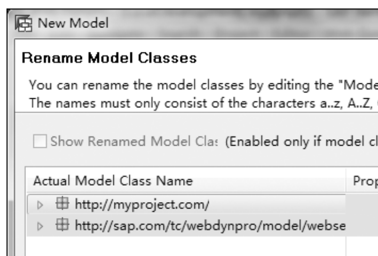


图 5-43　重命名导入类

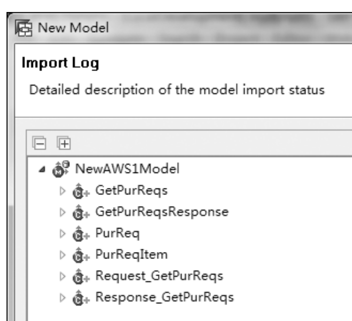


图 5-44　查看导入日志

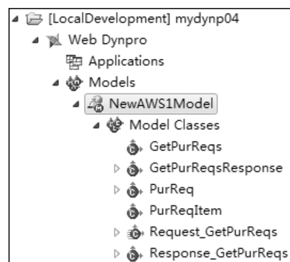


图 5-45　导入后的效果

10）在导入模型完成之后，需要创建对模型接口的调用，即创建自定义控制器。首先需要添加对导入模型的引用：右击组件中的“Used Models”，选择“Add”，如图 5-46 所示。

11）在弹出的对话框中选中之前导入的“NewAWS1Model”模型，并单击“OK”按钮，如图 5-47 所示。

12）添加引用之后，需要创建一个自定义控制器：在组件中右击“Custom Controller”，并选择“Apply Template”，如图 5-48 所示。

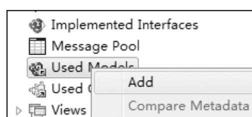


图 5-46　添加使用模型

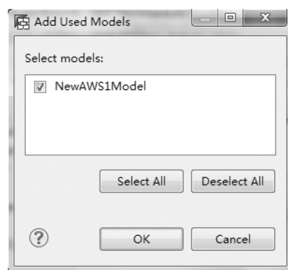


图 5-47　选择模型

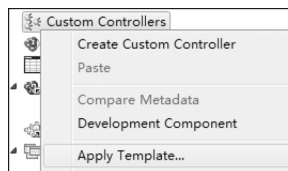


图 5-48　选择应用模板的菜单选择

13）在弹出的对话框中选择第一项“Service Controller”，并单击“Next”按钮，如图 5-49 所示。

14）选择之前导入的“NewAWS1Model”模型的请求方法，并单击“Next”按钮，如图 5-50 所示。

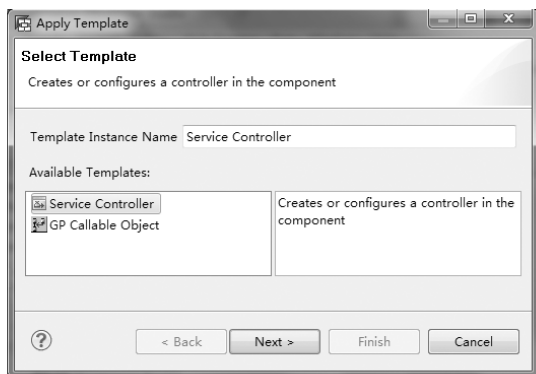


图 5-49　选择模板类型

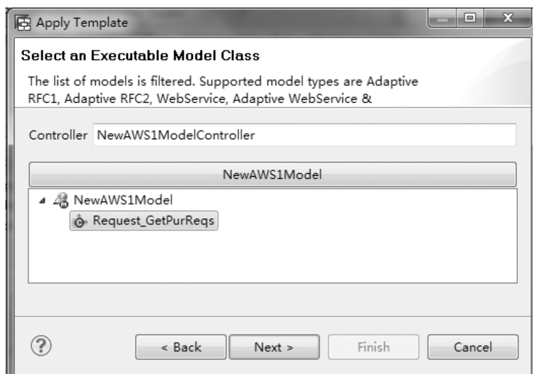


图 5-50　选择对应请求方法

15）选中所有的结点作为自定义控制器中的上下文并同时完成与模型的映射配置，随后单击“Next”按钮，如图5-51所示。

16）默认可以同时创建模型的执行方法，即触发接口调用，输入方法名称，随后单击“Finish”按钮，如图5-52所示。

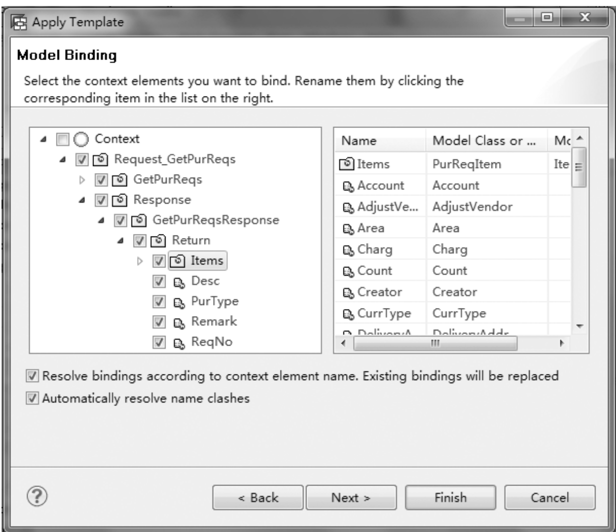


图5-51　选择上下文结点

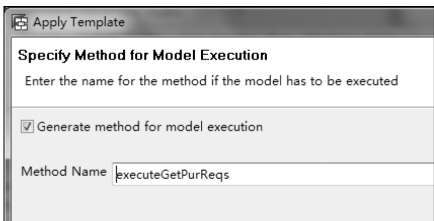


图5-52　输入执行方法名称

17）完成后保存所有修改，可以在模型编辑器中看到映射关系已创建，如图5-53所示。

18）打开组件控制器，并定义对应的上下文和调用方法，右击上下文中的“Context”结点，选择新建结点，如图5-54所示。

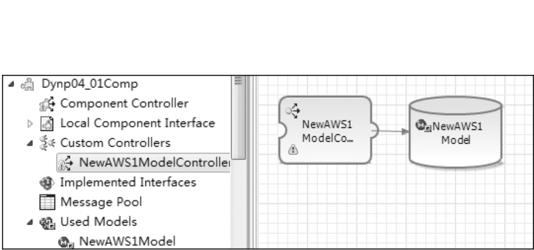


图5-53　添加自定义控制器后的效果

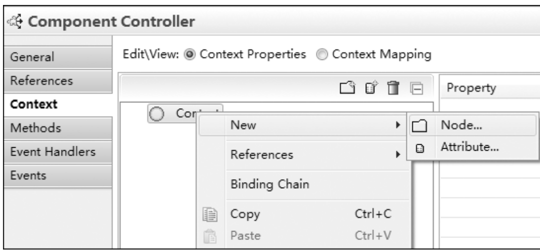


图5-54　创建上下文结点

19）在弹出的对话框中选中“Mapping”单选按钮，单击“Next”按钮，如图5-55所示。

20）选择创建的“NewAWS1ModelController”自定义控制器，随后选择对应的结点，单击“Next”按钮，如图5-56所示。

21）选中相应的结点和特性后，单击“Finish”按钮，如图5-57所示。

22）创建上下文完成后，保存所有修改即可，如图5-58所示。



图 5-55　选择结点类型

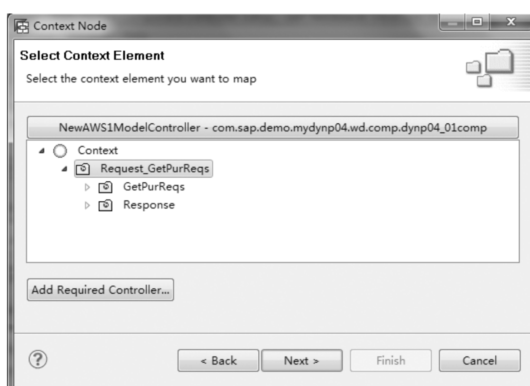


图 5-56　选择对应结点

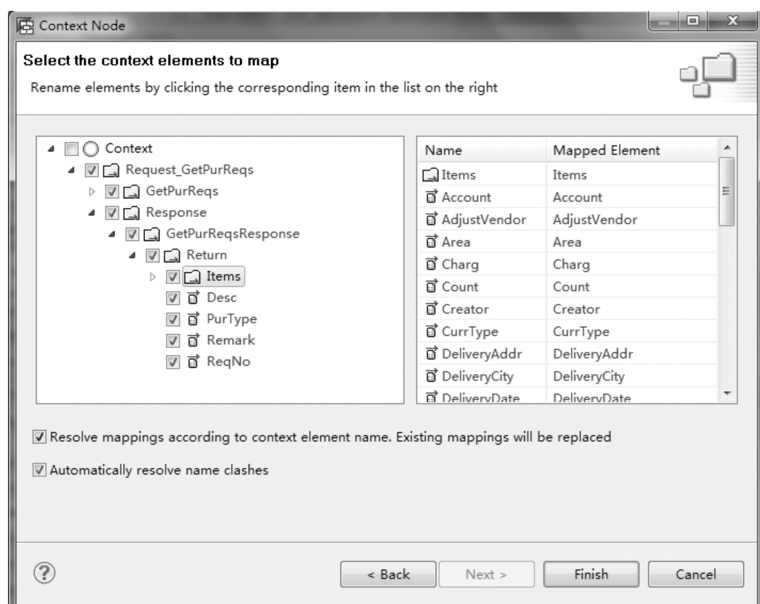


图 5-57　选择对应上下文

23）进入自定义方法的页签，单击加号创建一个自定义方法，如图 5-59 所示。

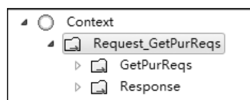


图 5-58　上下文清单

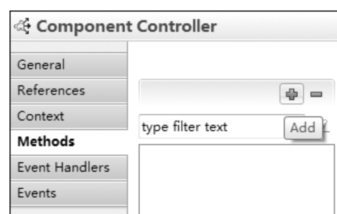


图 5-59　自定义方法页签

24）输入方法的名称，由于不需要配置参数，因此单击“Finish”按钮即可，如图 5-60 所示。

25）保存修改后，进入 Java 编辑器，在方法中添加代码（见图 5-61）：

```
wdThis. wdGetNewAWS1ModelControllerController( ). executeGetPurReqs( );
```

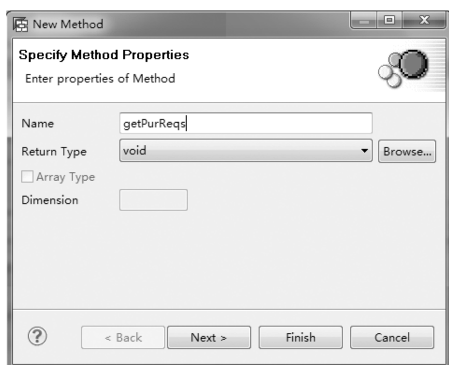


图 5-60　输入方法名称

```
public void getPurReqs( )  {
  //@@begin getPurReqs()
  wdThis.wdGetNewAWS1ModelControllerController().executeGetPurReqs();
  //@@end
}
```

图 5-61　添加代码

注： 对于简单架构的程序，可以跳过组件控制器中上下文和自定义方法的创建，直接从视图映射到自定义控制器即可，但还是推荐从组件控制器进行调用，方便后期运维管理。

26）进入视图，创建对应的上下文，右击上下文中的“Context”结点，选择新建结点，如图 5-62 所示。

27）在弹出的对话框中选择“Mapping”映射结点，随后单击“Next”按钮，如图 5-63 所示。

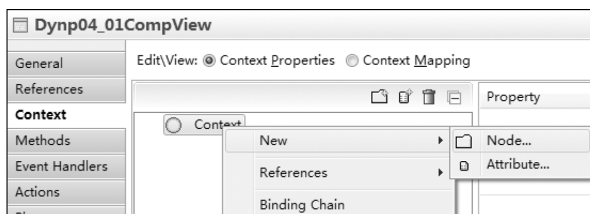


图 5-62　创建上下文结点

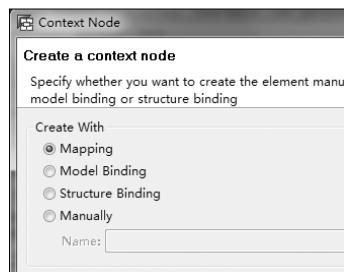


图 5-63　选择结点类型

28）选中需要进行映射的控制器，并单击“OK”按钮，如图 5-64 所示。

29）选中相应的结点和特性后，单击“Finish”按钮，如图 5-65 所示。

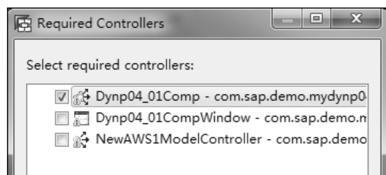


图 5-64　选择控制器

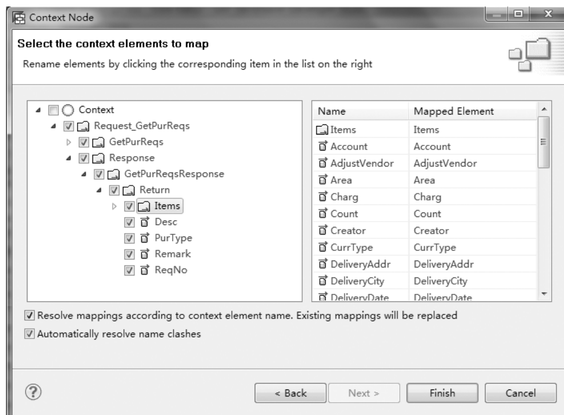


图 5-65　选择对应上下文

30）接着创建视图中的页面元素，右击“RootElement”，在弹出的快捷菜单中选择“Apply Template”，如图5-66所示。

31）选择“Form”，随后单击“Next”按钮，如图5-67所示。

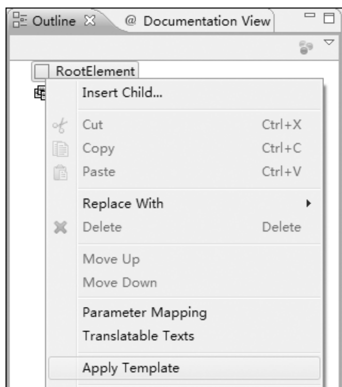


图5-66　应用模板的菜单选择

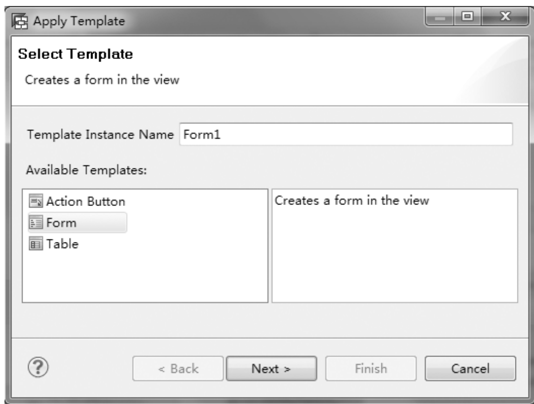


图5-67　选择模板类型

32）选中需要创建元素的上下文特性，单击“Next”按钮，如图5-68所示。

33）确认元素类型后，单击“Finish”按钮，完成查询条件的页面元素创建，如图5-68所示。

34）接着创建一个查询按钮，右击透明容器，选择“Insert Child”，如图5-70所示。

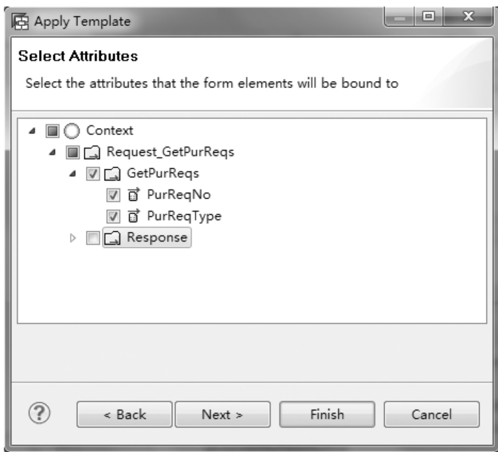


图5-68　选择对应上下文

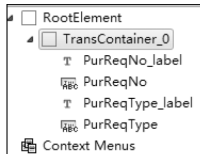


图5-69　创建后的元素清单

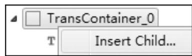


图5-70　插入子元素的菜单选择

35）选择“Button”按钮类型，单击“OK”按钮，如图5-71所示。

36）完成按钮创建后，创建默认动作，切换到事件页签并单击“Create”按钮，如图5-72所示。

37）输入动作名称“Seach”，单击“Finish”按钮，如图5-73所示。

38）保存修改后，进入Java编辑器，在按钮处理方法中添加代码（见图5-74）：

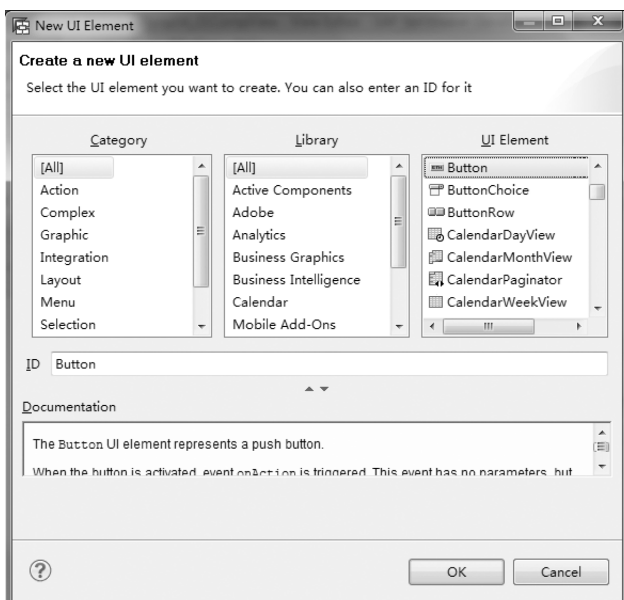


图 5-71　选择元素类型

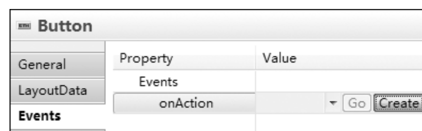


图 5-72　修改事件属性

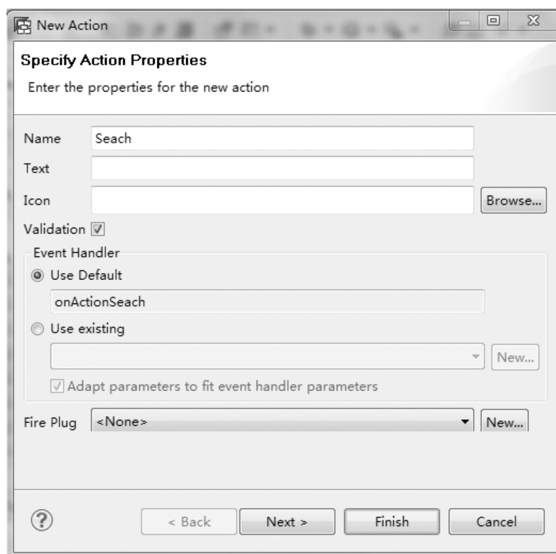


图 5-73　输入动作名称

```
wdThis. wdGetDynp04_01CompController( ). getPurReqs( ) ;
```

```
public void onActionSearch(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionSearch(ServerEvent)
  wdThis.wdGetDynp04_01CompController().getPurReqs();
  //@@end
}
```

图 5-74　添加代码

39）接着再次应用模板，创建采购申请抬头字段的返回数据，右击“RootElement”，在弹出的快捷菜单中选择“Apply Template”，如图 5-75 所示。

40）选择“Form”，单击“Next”按钮，如图 5-76 所示。

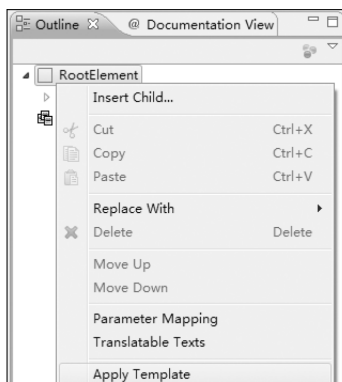


图 5-75　应用模板的菜单选择

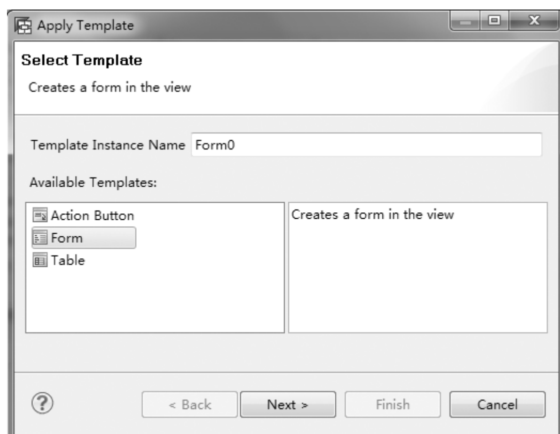


图 5-76　选择模板

41）选中采购申请的必要信息，单击“Next”按钮，如图 5-77 所示。

42）将其中的采购申请描述和采购申请备注两个字段（即“Desc”和“Remark”字段）修改为“TextEdit”视图元素类型，随后单击“Finish”按钮完成表单创建，如图 5-78 所示。

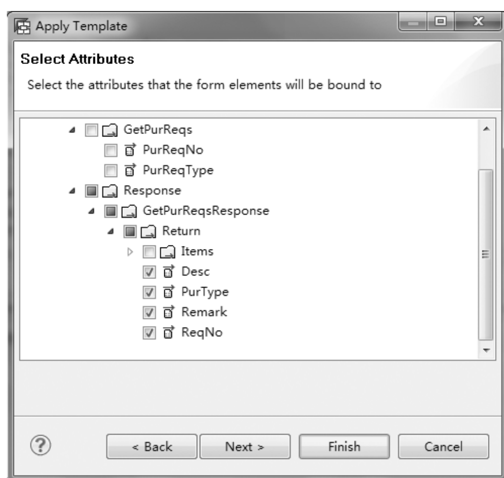


图 5-77　选择对应结点

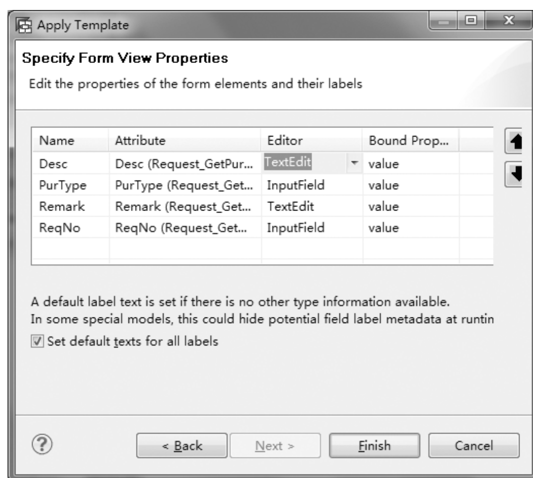


图 5-78　调整元素类型和位置

43）创建完成后，保存所有修改，如图 5-79 所示。



图 5-79　预览效果

44）再次应用模板，创建采购申请行项目的表元素，右击“RootElement”，在弹出的快捷菜单中选择“Apply Template”，如图 5-80 所示。

45）选择“Table”，单击“Next”按钮，如图 5-81 所示。

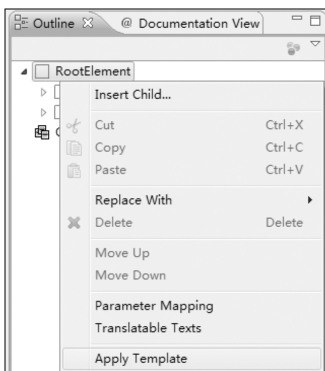


图 5-80　应用目标的菜单选择

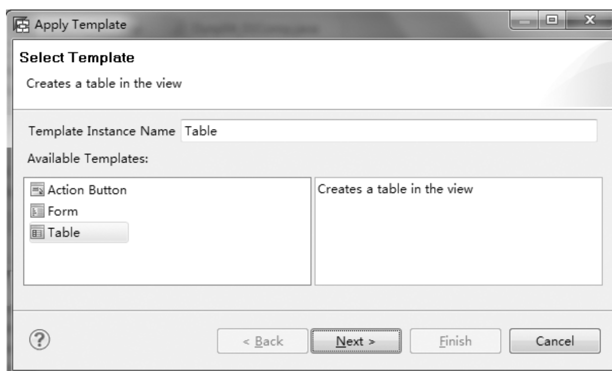


图 5-81　选择模板类型

46）由于行项目中字段较多，如全部在表格中显示，会使得页面宽度超出正常页面，不便于用户查看详细的数据信息，而且参考 ECC 系统中的采购申请界面，因此仅在表元素中放入部分字段，剩余字段都用选项卡的方式在表格下方进行显示，如图 5-82 所示。

47）可单击右侧的上、下箭头按钮调整不同字段的排列顺序，也可在完成创建后从“Outline”清单中进行调整，调整完毕后单击“Finish”按钮，如图 5-83 所示。

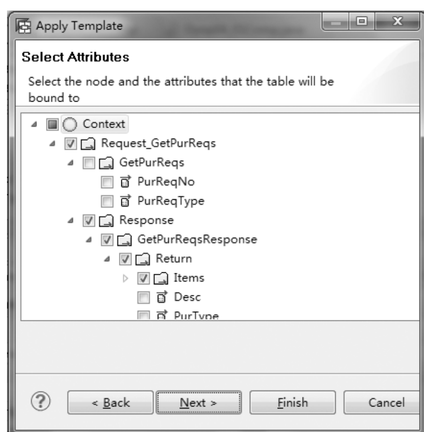


图 5-82　选择对应上下文

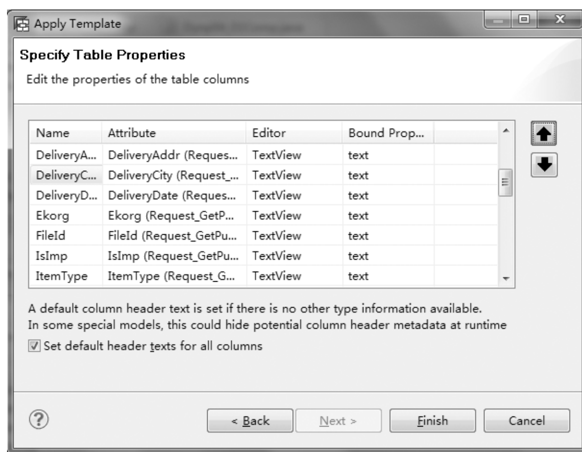


图 5-83　调整元素的类型和位置

48）保存所有修改，完成了查询输入字段元素、返回的采购申请信息元素以及行项目信息元素的创建，如图 5-84 所示。

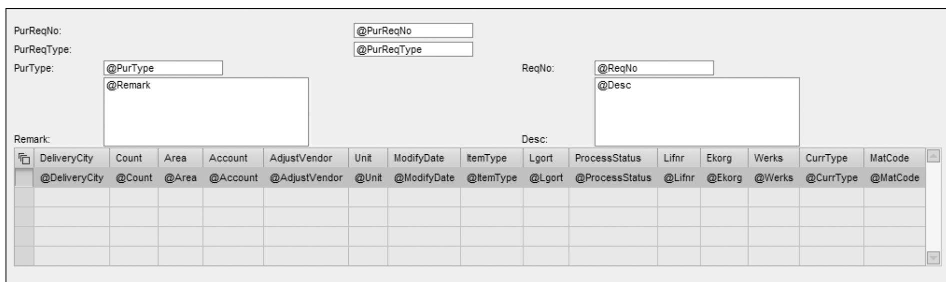


图 5-84　视图预览效果

49）接着创建选项卡集，将其他采购申请字段放入其中进行显示，右击“RootElement”，在弹出的快捷菜单中选择“Insert Child”，如图 5-85 所示。

50）选择“TabStrip”，单击“OK”按钮，如图 5-86 所示。

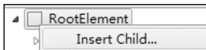


图 5-85　插入子元素的菜单选择

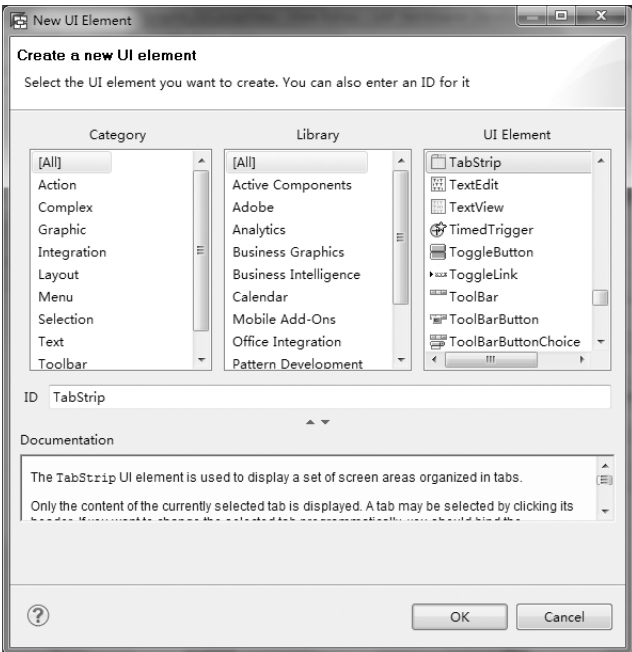


图 5-86　选择元素类型

51）接着右击“TabStrip”，选择“Insert Tab”，如图 5-87 所示。

52）重复添加选项卡操作，共添加 3 个选项卡，如图 5-88 所示。

53）将 3 个选项卡的“Tab_Header”标题分别改为“物料数据”“联系人”和“客户数据”，如图 5-89 所示。

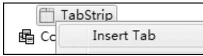


图 5-87　插入子元素的菜单选择

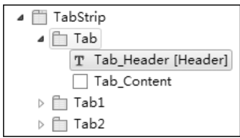


图 5-88　创建后的元素清单



图 5-89　运行效果

54）随后右击选项卡中的透明容器“Tab_Content”，选择“Apply Template”，如图 5-90 所示。

55）在弹出的对话框中选择“Form”，单击“Next”按钮，如图 5-91 所示。

56）选择需要在该选项卡中显示的字段，单击“Finish”按钮，如图 5-92 所示。

57）完成“物料数据”选项卡的元素创建后，保存所有修改，如图 5-93 所示。

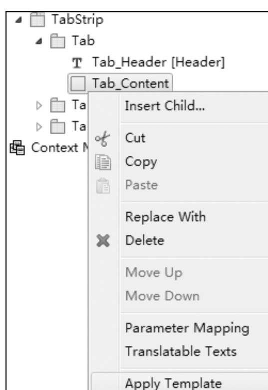


图 5-90　应用目标的菜单选择

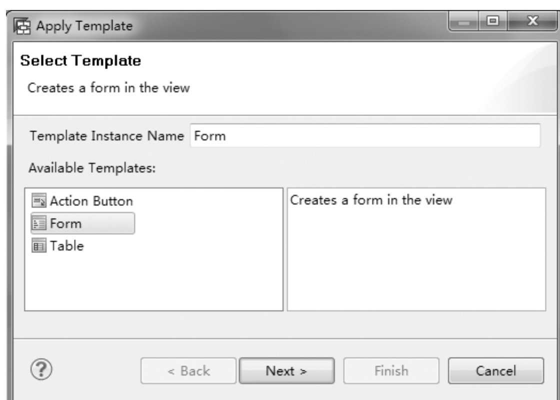


图 5-91　选择模板类型

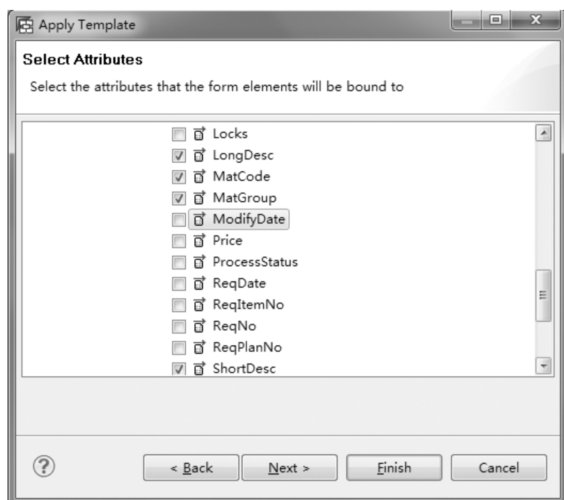


图 5-92　选择对应上下文



图 5-93　“物料数据”的预览效果

58）重复该选项卡的操作，接着创建“联系人”选项卡中的元素，如图 5-94 所示。

59）重复上述操作，接着创建“客户数据”选项卡中的元素，如图 5-95 所示。



图 5-94　“联系人”的预览效果



图 5-95　“客户数据”的预览效果

60）完成所有配置后，页面的雏形已基本完成，接着创建应用程序进行发布，右击“Applications”，选择“Create Application”，如图 5-96 所示。

61）输入应用程序的名称，随后单击“Next”按钮，如图 5-97 所示。

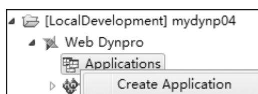


图 5-96　创建应用的菜单选择

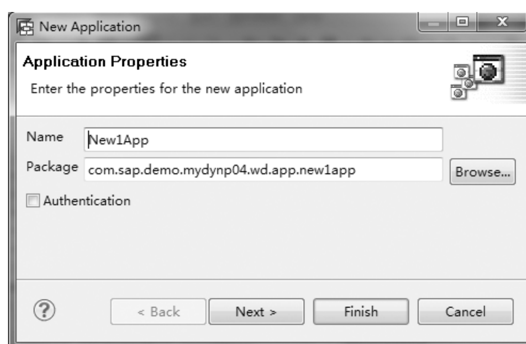


图 5-97　输入应用名称

62）选择使用已存在的组件，单击“Next”按钮，如图 5-98 所示。

63）默认选择已完成配置的组件和接口视图，单击“Finish”按钮，如图 5-99 所示。

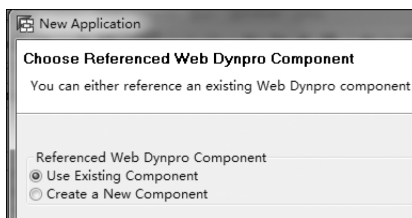


图 5-98　选择组件类型

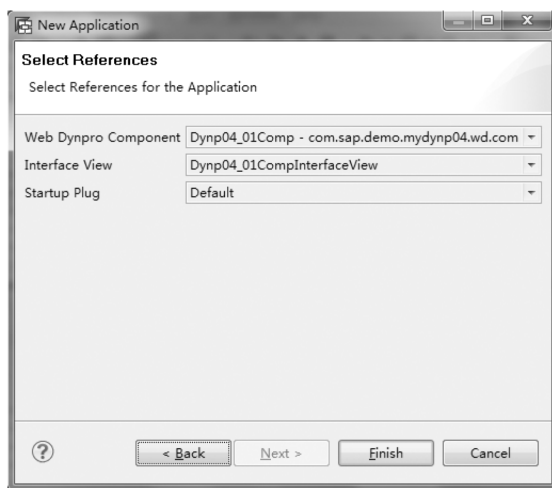


图 5-99　选择对应引用

64）保存所有修改，右击应用程序，选择“Deploy New Achive and Run”，如图 5-100 所示。

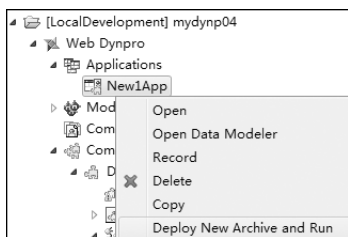


图 5-100　部署和运行应用的菜单选择

65）运行后即可对程序功能进行测试，单击“查询”按钮可以得到 Web 服务中预设的返回值，如图 5-101 所示。

图 5-101　预览效果

5.2.3　视图元素调整

在项目开发过程中，功能测试通过标志着代码部分工作已基本完成。本节将对之前完成的页面进行布局和位置的优化。

开发实例：

1）优化查询表单，采用 3＊30 界面设计。即每一行有 3 个属性，8 个元素，每个元素 General Width 为 100%，通过控制 Layout Width 来控制页面布局。每个属性的 Label 与 TextValue 分别占 10% 与 20%，剩下的 20% 通过添加 HorizontalGutter 来补足。

2）首先选中查询条件的透明容器（第一个 TransContainer），将它一般属性中的“Layout”布局修改为“GridLayout”，随后进入布局属性，将“colCount”改为“8”，如图 5-102 所示。

3）在“Outline”视图中批量选中所有的页面元素，如图 5-103 所示。

图 5-102　属性修改

图 5-103　元素清单

4）将它们的“width”属性设置为“100%”，如图 5-104 所示。

图 5-104　属性修改

注：批量选中视图元素时，属性页签中显示的是这些元素的共有属性，修改属性值会应用至所有选中的视图元素。

5）按住〈Ctrl〉键批量选中两个“lable”标签元素，将它们的“width”属性设置为

“10%”，如图 5-105 所示。

6）同样地，批量选中两个“InputField”输入框元素，将它们的“width”属性设置为“20%”，如图 5-106 所示。

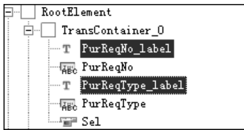


图 5-105　元素清单 1

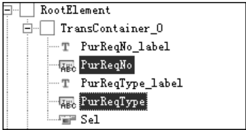


图 5-106　元素清单 2

7）插入“HorizontalGutter”分割线，并在布局属性中将“width”改为“5%”，将一般属性中的“visible”改为“blank”空白。

8）随后将两个标签的“text”属性分别修改为“采购申请号”和“申请类型”，保存所有修改，如图 5-107 所示。

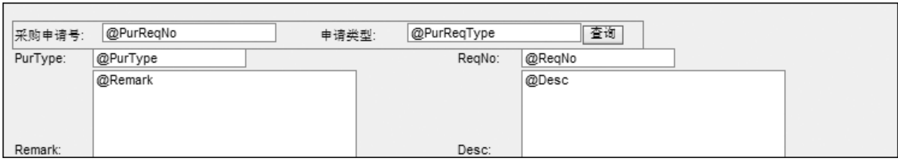


图 5-107　页面预览效果

9）接下来优化返回信息的页面元素，首先添加“Group”，并将返回元素的透明容器、表和选项卡集拖曳至新建的组中，如图 5-108 所示。

10）修改组的抬头文本描述，将“text”属性修改为“采购申请返回”。

11）按照步骤 1）～6），以同样的方式返回信息的透明容器“TransContainer_0_0”，如图 5-109 所示。

12）将“Remark_label”和“Desc_label”的布局属性中的“vAlign”改为“top”，即靠近文本框的顶部，如图 5-110 所示。

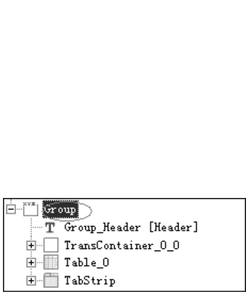


图 5-108　元素清单 3

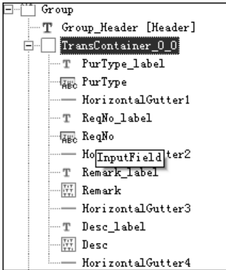


图 5-109　元素清单 4

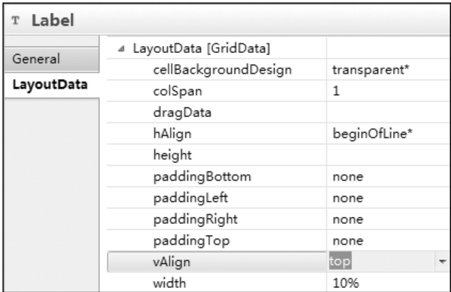


图 5-110　修改属性

13）随后将 4 个“label”标签的“text”文本属性分别修改为“申请类型”“备注”“采购申请”和“描述”，随后保存所有修改，如图 5-111 所示。

14）对于表格元素，将所有列的“Header”标题修改为对应的中文描述，如图 5-112 所示。

采购申请返回

申请类型: @PurType　　采购申请: @ReqNo

备注: @Remark　　描述: @Desc

图 5-111　视图预览效果 1

行项目

交货城市	数量	区域	科目	推荐供应商	单位	修改日期	行项目类别	库存地点	处理状态	供应商	采购组织	工厂	现金类型	物料编码
@DeliveryCity	@Count	@Area	@Account	@AdjustVendor	@Unit	@ModifyDate	@ItemType	@Lgort	@ProcessStatus	@Lifnr	@Ekorg	@Werks	@CurrType	@MatCode
@DeliveryCity	@Count	@Area	@Account	@AdjustVendor	@Unit	@ModifyDate	@ItemType	@Lgort	@ProcessStatus	@Lifnr	@Ekorg	@Werks	@CurrType	@MatCode
@DeliveryCity	@Count	@Area	@Account	@AdjustVendor	@Unit	@ModifyDate	@ItemType	@Lgort	@ProcessStatus	@Lifnr	@Ekorg	@Werks	@CurrType	@MatCode

图 5-112　视图预览效果 2

15）最后根据实际需要，调整个别元素的实际宽度和位置，保存所有修改，并重新编译、运行，可以看到优化后的页面，如图 5-113 所示。

采购申请号:　　申请类型:　　查询

采购申请返回

申请类型: 01　　采购申请: 010000010000

备注: 采购申请备注　　描述: 采购申请描述

行项目

交货城市	数量	区域	科目	推荐供应商	单位	修改日期	行项目类别	库存地点	处理状态	供应商
南京	10	CN	1601030000	10000000001	EA	20120908	01	Z001	A01	0101000020

物料数据　联系人　客户数据

短描述: 电缆\00000065069548\液压支架电液控制阀\DBT\进口

长描述: 电缆\00000065069548\液压支架电液控制阀\DBT\进口

单位: EA

物料组: G001

物料编码: 000000000010000315

图 5-113　运行效果

注：在实施项目过程中，一方面界面需要参考用户原始系统中的操作习惯以及界面风格，另一方面也要务必与业务顾问及用户进行确认。

5.3　延伸配置

1. SAP 企业门户（Portal）

在 SAP NetWeaver 架构中，企业门户既是用户访问系统的统一入口，也是 WD4J 应用程序默认的应用系统。在 Web Dynpro 应用程序开发完成后，需要进一步在企业门户中将应用程序以门户视图的形式添加至菜单和角色中。在完成门户内容配置之后，用户在登录企业门户后也就能访问与他角色权限相符合的页面菜单以及应用程序清单。

至于权限控制的概念，在 Web Dynpro 中，可以通过代码或者上下文绑定来控制不同角色用户看到个性化的应用页面，甚至限制使用应用程序中的部分功能，所以程序内的权限控

制都是在 Web Dynpro 中完成；在企业门户中，则是将整个 Web Dynpro 项目作为一个对象进行添加和分配，所以对于企业门户中的权限控制决定了用户是否能看到对应应用程序的访问链接。开发人员需要了解并分清属于开发控制内容，还是属于门户配置控制内容。

2. SAP 业务流程管理（BPM）

如果企业门户的目标是将所有系统的入口统一起来，那么 SAP 业务流程管理的目标就是将所有系统的业务流程也统一到一个平台中。

SAP 业务流程管理作为 SAP NetWeaver 平台的重要组成部分，提供了设计、业务流程建模、实施、运行、流程运行监控的一整套工具，帮助 SAP 的客户机与 SAP 已有的应用模块实现自动化的业务流程，跨越不同的系统来控制和提升业务流程的运行效率。

除了业务流程需要单独的 BPM 流程驱动引擎外，所有 BPM 的页面相关内容和业务数据处理逻辑均通过 WD4J 技术实现。

第 6 章　SAP 企业门户

企业门户以其特有的系统入口统一性、多平台视图页面的集成性以及日益增长的企业信息化对于门户的依赖性，正逐渐成为企业系统信息化的重要环节。本章介绍 SAP 企业门户的一些关键内容，以及企业门户与 Web Dynpro 开发相关的配置，当然也同样会提供具体的开发实例供用户参考。

6.1　门户与企业门户

6.1.1　门户

在 20 世纪末，互联网兴起、门户萌芽时期，网站还都处于基础的页面展示阶段，以国内门户网站为例，大多以新闻信息为主要内容，而这样的门户只能算是一个信息门户，而并非企业真正需要的门户。在不断地优化升级后，门户已经从简单的网页门户，到企业信息平台，再到企业应用集成，最终成为了一个用户可以高效协作的平台。当然，如今的企业门户也是一个提供给用户单点登录不同业务系统、优化业务流程并且可以无缝地将各种企业应用整合至浏览器端界面的平台。

从个人角度来看，SAP 的企业门户为所有的企业应用提供了一个统一的入口，弱化了企业中各种各样的“系统”，为不同系统中的业务进行串联和整合提供了基础，并将在企业门户平台上开发的功能成为企业门户中的“应用”。在今天 SAP 的 HANA 云平台大行其道的时候，从某个角度来看，企业门户也可以算是为企业用户提供了一个“私有云”。两者的不同之处在于，企业门户的出现是因为先有了不同的业务系统、复杂的业务流程和多样化的数据存储，后出现了 NetWeaver 的架构从不同层次进行整合，也就是想把老积木放进一个沙盘中；而 SAP 的 HANA 云平台从一开始就想要把原来的积木推倒重来，随后再根据不同企业的业务、不同行业的标准以及不同客户的需求具体实施细化的解决方案（如基于 Fiori 的 App 应用），也就是提供了一个更大的沙盘，不仅可以放下以前的积木，还会提供更多的新应用。

6.1.2　SAP 企业门户

SAP 企业门户（SAP Enterprise Portal，SAP EP）是一个包含了知识管理和协作服务的复杂门户架构，基于 SAP 封装中的 Web Dynpro For Java 语言进行开发实现，系统中有多样化的业务内容和组件，用于保证系统的快速实施以及方便用户使用。它的核心功能有以下几个方面。

- 门户架构：提供了 SAP 企业门户的基本功能，并且保证基于浏览器的多种信息访问，集成了应用、信息和流程等不同内容。
- 协作：提供给使用者可以进行高效协作的一个平台，可以实现虚拟聊天室、基础论坛等功能，而且也可以与共享电子邮件、日历等控件进行集成。
- 知识管理：可以对企业的非结构化数据进行管理。在 SAP ERP 的生态中，结构化数据管理有对应的数据仓库（BW）系统，而非结构化的数据就通过门户中的“知识管

理”功能来进行存储和访问。

SAP 企业门户为了降低总体拥有成本（TCO）而做了很多的努力，从系统的后台配置到前端的用户体验，门户都提供了一些相关的工具。

- 多平台支持：当用户部署程序（并不限于 Web Dynpro 程序）到 SAP NetWeaver 应用服务器时，支持不同的操作系统环境，不论是 UNIX 系统还是微软的 Windows 服务器，平台都有成熟的监控管理机制。
- 业务包：除了 SAP 企业门户本身自带的一些内容，它还允许部署和运行第三方的业务包进行功能扩展。业务包包含了一些预定义的门户内容，可以方便地集成到门户当中。
- 多语言支持：为了适应跨地区、跨国家的多语言公司，SAP 企业门户提供了超过 20 种语言进行配置，并可以通过语言配置向导实现自动翻译。
- 远程门户网络：如果企业内有不同版本的 SAP 企业门户，或者需要将两个或多个企业门户联合起来进行访问，就可以通过远程门户网络的配置，提供用户统一入口的无缝集成体验。

访问 SAP 企业门户默认的地址是：http://<ServerHost>:<ServerPort>/irj，随后在登录界面输入对应的用户名和密码即可登录（其中，服务器域名由局域网的 DNS 域名服务器决定，而服务器端口可以在系统中进行自定义配置）。在使用默认的管理员账号“Administrator”登录后，可以看到企业门户的默认页面和框架，最上方包含企业 Logo、搜索栏和相关链接，第二行是系统页面菜单，下方的文件夹则是企业门户的角色对应的顶层目录，下方则是页面的具体内容显示，如图 6-1 所示（该图以 NetWeaver 7.4 的企业门户为例，而门户整体页面的框架、链接索引都可以通过系统的主题和界面配置进行个性化。



图 6-1　SAP 企业门户登录后的页面

拥有管理员权限的用户在登录后就能看到企业门户的“内容管理”“用户管理”和“系统管理”三大块配置内容。下面重点介绍 SAP 企业门户中与 Web Dynpro 开发相关的一些配置内容。

6.2 内容管理

6.2.1 门户内容介绍

SAP 企业门户的内容包括了门户中所有的标准内容（角色、页面等）以及通过配置添加自定义的对象，而这些对象主要有 4 种类型，见表 6－1。

表 6－1　门户对象的类型

门户内容	定　义
iView 门户视图	门户视图是门户内容中最基本的单位，它支持从不同的系统获取信息以进行展示，如果将门户想象为一张报纸，那 iView 就是组成门户的那些文章。企业门户中 iView 的优势在于支持从不同的系统获取信息以进行展示，包括 ERP 系统（企业资源管理系统）、CRM 系统（客户关系管理系统）、网页以及 BW 报表（数据仓库系统）等不同架构
Page 页面	一个页面可以包含一个或多个 iView，或者其他的页面，它们依照一定的布局进行排列。页面可以被分配给工作集或者角色，在用户调用该页面时就可以显示相应的内容
Workset 工作集	工作集将门户视图和页面以树状结构的形式组织在同一个目录中，工作集本身则作为一种通用的、可重用的内容结构提供给角色
Role 角色	角色是内容实体中最高级的实体，所有的其他实体都需要分配给角色才能展示给用户。角色同样也代表企业门户系统中的页面链接查看权限，视图、页面和工作集都不能分配给用户，只有角色可以被分配给用户

一般来说，角色的分配会存在一个“角色－工作集－页面－门户视图”这样一个递进的关系，但在实际配置中，它们之间的关系可以非常灵活，如图 6-2 所示。由图 6-2 可以看到，最右侧是零散的视图和页面，其中红色方块表示门户视图，蓝色方块表示页面，它们既可以分配给工作集，也可以跳过工作集直接分配给角色；而同样的情况，工作集也可以分配给另一个工作集，角色也可以分配给另一个角色。在整个角色的树状结构中，其实不管是角色或是工作集，扮演的都是一个文件夹的作用，所以图中也使用了文件夹的图标进行表示。唯一要牢记的是，只有角色可以分配给用户。

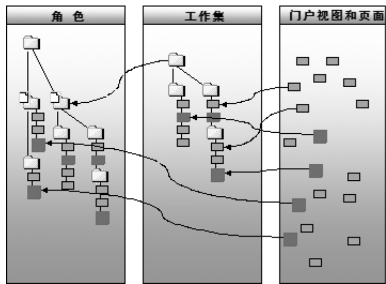


图 6-2　门户对象之间的关系

对于 Web Dynpro 开发的应用程序，运行时都会以网页的形式进行展现，并且每一个应用程序都有一个独特的 URL 链接。对于用户来说，并不需要拿到一个链接清单进行一一的访问，这时就需要企业门户统一地将 Web Dynpro 开发的应用通过门户视图进行集成并以菜单形式展现。在内容管理中，将会完成页面的配置、菜单的定义以及角色的创建与分配。

在实际的项目应用中，一般会把所有开发的应用复制到一个统一的项目文件夹下，以便于管理和维护；然后针对不同的功能或分类，建立不同的子文件夹以及菜单所属的角色，并将应用与角色关联在一起，最后利用页面包含、工作集归类、角色分配以及内容对象的互相包含完成整个门户应用的配置。

下面将详细地介绍如何将 Web Dynpro 程序通过 SAP 企业门户中页面角色的配置向用户进行展示（截图都是以新版本的门户进行展示，界面与老版本的 EP 7.0 界面稍有不同）。

6.2.2 门户页面创建

从逻辑角度来说，Web Dynpro 项目（EAR 包）的部署其实仅仅是将程序发布到了 NetWeaver 的 Java 应用服务器上。虽然一般情况下这个服务器也同时是 SAP 企业门户的服务器，但 Web Dynpro 程序并没有在门户中存在一个真正的对象，这就需要进行以下的配置操作。

配置实例：

1）首先使用有管理员权限的账户登录门户系统，在默认的内容管理页面下，可以看到若干系统自带的文件夹目录，如图 6-3 所示。

2）单击选中“门户内容”，随后右击并选择“新建”→“文件夹”，如图 6-4 所示。



图 6-3　门户内容中默认的目录结构



图 6-4　创建文件夹的菜单选择

3）在右侧显示的创建文件夹向导中，在对象名称文本框中输入“项目内容”，在“对象标识”文本框中输入“proj”，随后单击“下一步”按钮，如图 6-5 所示。

4）确认文件夹相关信息填写正确，随后单击“完成”按钮完成创建。创建这样一个自定义文件夹的目的是为了更好地进行门户内容的管理，后续相关的内容都会放入这个文件夹，方便了维护管理以及最终上线的传输工作，如图 6-6 所示。



图 6-5　输入文件夹信息



图 6-6　确认文件夹信息

5）接着在目录中找到“Web Dynpro Java 应用程序”并展开目录，这个结点下显示了服务器已经成功部署的所有 WebDynpro 工程，包括所有的标准应用和自定义开发程序，找到需要进行显示的 Web Dynpro 应用程序的名称，如图 6-7 所示。

6）展开 Web Dynpro 工程名称，会出现对应工程所包含的应用程序（有可能会存在多个应用程序），单击选中应用程序，如图 6-8 所示。

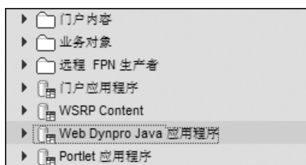


图 6-7　Web Dynpro 应用程序对应目录

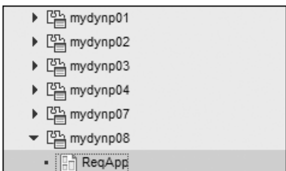


图 6-8　选中 Web Dynpro 应用程序

7）随后右击该应用程序，在弹出的快捷菜单中选择“复制”，如图 6-9 所示。

8）返回最开始创建的自定义文件夹“项目内容”，右击文件夹，如图 6-10 所示。

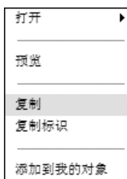


图 6-9　操作菜单

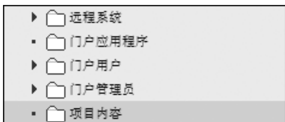


图 6-10　门户目录中自定义文件夹

9）在弹出快捷菜单中选择“粘贴作为 PCD 对象”，如果之前的复制动作没有成功完成，这里的快捷菜单中就不会出现“粘贴作为 PCD 对象”的菜单选项，如图 6-11 所示。

10）在右侧显示的创建页面的向导中，在对象名称文本框中输入“请假申请”，在对象标识文本框中输入“ReqApp”，随后单击“下一步”按钮，如图 6-12 所示。

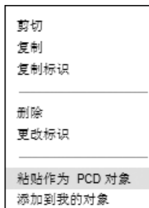


图 6-11　粘贴对象的菜单选择

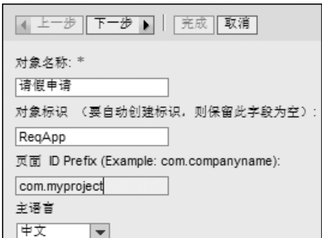


图 6-12　输入对象信息

11）确认页面相关信息填写正确，单击“完成”按钮完成创建，如图 6-13 所示。

12）右击该页面，在弹出的快捷菜单中选择“预览”，如图 6-14 所示。

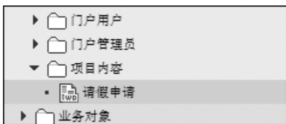


图 6-13　创建后的效果

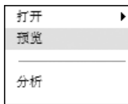


图 6-14　操作菜单

13）Web Dynpro 程序将会在新窗口中运行，可以对页面进行预览，如图 6-15 所示。

完成上述配置后，一个 Web Dynpro 程序才从 Java 应用服务器上的“程序”真正地集成到了一个企业门户的“页面”中。



图 6-15　运行效果

6.2.3　门户角色的创建和分配

通过角色权限的配置，就可以实现针对不同用户展示不同的内容。例如，业务操作员可以在门户中看到自己使用的业务操作界面，而他却不能看到业务分析报表；企业管理层的门户可以展示业务分析报表，而他却不需要展示业务操作界面。

在 SAP 企业门户中，页面无法直接分配给用户，必须经过“角色”的封装和管理，才能赋予用户进行展示。门户的角色在内容开发工作室中创建，角色中除了包含页面对象之外，还需要设定内容在门户中的导航结构，即在门户中的访问菜单和路径。

开发实例：

1）返回最开始创建的自定义文件夹“项目内容”，右击文件夹，在弹出的快捷菜单中选择“新建”→“角色”→“自由式角色”，如图 6-16 所示。

2）在右侧显示的创建角色向导中，在“对象名称”“对象标识”和“角色 ID Prefix”文本框中分别输入“人事管理”“PR_HRMNG”和“com. myproject”，随后单击“下一步”按钮，如图 6-17 所示。

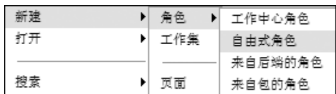


图 6-16　创建角色的菜单选择

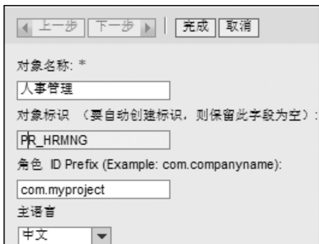


图 6-17　输入角色信息

3）确认角色相关信息填写正确，随后单击“完成”按钮完成创建，在文件夹中即会出现所创建的对象，如图 6-18 所示。

4）接着选中新建的角色，右击并选择“打开”→“角色”，如图 6-19 所示。

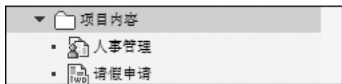


图 6-18　创建后的目录

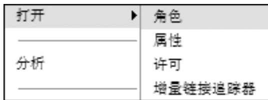


图 6-19　操作菜单

5）接着选中之前创建的页面，右击并选择“将页面添加到角色”→“增量链接”，如图 6-20 所示。如果未提前打开角色，右击页面后不会出现“将页面添加到角色”的选项。这里添加的两种方式“增量链接”和“复制”，可以分别理解为创建了一个快捷方式以及复制创建了一个新的对象。使用“增量链接”的添加方式，如果原始页面属性发生

变化，则角色中的对应增量链接也会跟随着同步改变；如果使用“复制”方式进行添加，则不会同步改变。

6）在添加完成后，在右侧的角色内容显示中，就能看到角色的一个树状结构，选中角色右侧的“入口点”，完成角色配置。入口点是在门户中显示的一级主菜单，也是通过门户进入子页面的切入点，选择角色为入口点，门户会以角色内容为一级菜单依次显示导航结构菜单，如图 6-21 所示。

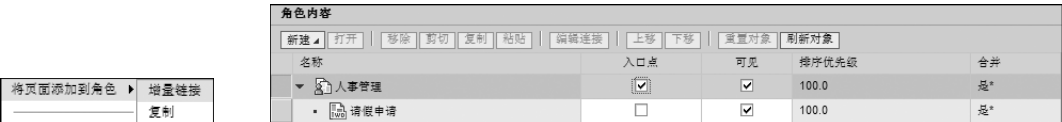


图 6-20　操作菜单　　　　图 6-21　角色配置

7）右侧可以看到导航结构的预览（如果仅勾选“请假申请”页面，则在预览中也仅会显示“请假申请”一级菜单），如图 6-22 所示。

8）在完成了角色分配的操作后，最终需要将角色分配给指定的用户，该用户登录后才能正确地看到为他所配置的 Web Dynpro 页面。首先进入用户管理，创建一个测试用户，单击“创建用户”按钮，如图 6-23 所示。

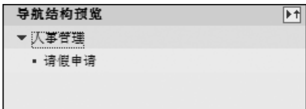


图 6-22　预览效果

9）输入登录标识（用户 ID）、定义密码、确认密码、姓氏和名字，如图 6-24 所示。

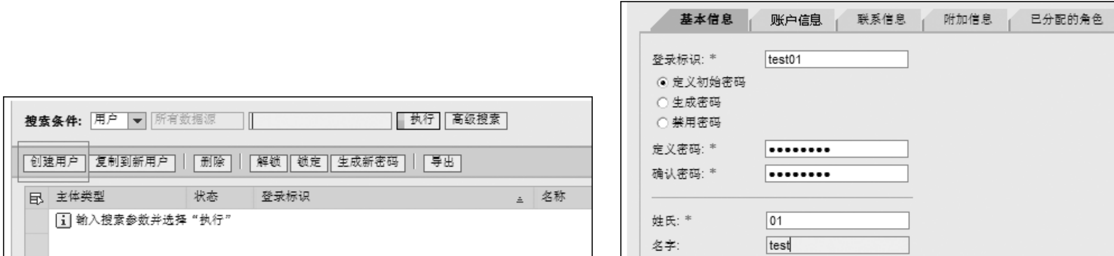


图 6-23　搜索用户界面　　　　图 6-24　输入用户信息

10）填写完基本信息后，切换到“已分配的角色”页签（图 6-24 最右侧页签），页签中会出现左右两个表格，左侧“可用角色”表示门户中的角色，右侧的“分配的角色”表示已分配给当前用户的角色，如图 6-25 所示。



图 6-25　修改角色界面

11）在左侧输入需要搜索的角色关键字，此处输入“ * HRMNG * ”单击“执行”按钮，下方会显示搜索结构，选中该角色后，单击下方的“添加”按钮，如图 6-26 所示。

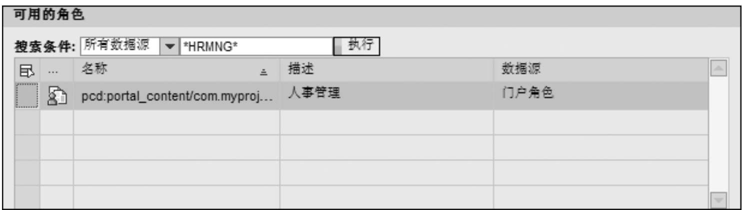


图 6-26　搜索可用角色

12）添加完成后，角色会出现在右侧的“已分配的角色”中，表示角色分配完成，随后单击“保存”按钮完成用户的创建和角色分配，如图 6-27 所示。



图 6-27　查看已分配角色

13）注销门户系统，使用刚才创建的“test01”登录后（首次登录可能需要修改密码），就能看到实际配置的 Web Dynpro 的页面，目录结构与在角色中的预览结构相同，如图 6-28 所示。



图 6-28　运行效果

6.2.4　多级菜单配置

在完成上述的配置后，了解到门户页面的简单配置过程包括页面创建和角色分配。当然在实际项目实施中很多复杂的功能模块会对应于多个门户入口，不同菜单下又会有不同的子菜单，同时要求不同组织架构的人能够分配不同的功能权限或子功能权限，这就需要在角色配置中实现多级菜单的展示。

在门户中多级菜单配置主要有以下 3 种方式：①在角色中创建文件夹（不是企业门户中的

文件夹），通过文件夹的层次结构来实现多级菜单；②额外创建门户对象“工作集”，通过工作集实现多级菜单并分配给角色进行展示；③多个角色的互相嵌套，从而实现多级菜单。

第一种方式，在角色中直接创建文件夹的效果更为直接和明显，适用于简单的菜单结构；另外两种方式则适用于多角色，但角色与角色之间同时存在个别差异的情况。例如，财务部门用户和人力资源部门用户有各自不同的业务处理页面，但他们又同时可以查询几十、上百张的公司基本信息报表，在门户配置中，不可能为财务部角色和人力资源部角色分别创建内部文件夹进行报表页面添加，而是将这些基本信息报表统一汇总至一个工作集或角色中，再进行具体的管理和分配。而具体是使用工作集进行管理，还是使用角色进行管理，就需要根据项目情况进行评估，如果不需要有额外的子角色，建议还是使用工作集进行管理，可以尽量地减少后期用户、角色的维护管理工作。

接着将通过一个实例来具体分析两种配置方式，模拟的多级菜单见表 6－2。

表 6－2　模拟的多级菜单

一 级 菜 单	二 级 菜 单	三 级 菜 单
人事管理	考勤	考勤录入
		（其他子菜单……）

1. 角色内配置

1）首先选中新建的角色，右击并选择“打开”→“角色”，如图 6-29 所示。

2）在右侧的角色内容中，单击“新建”按钮，如图 6-30 所示。

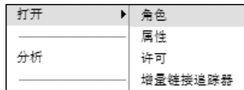


图 6-29　操作菜单

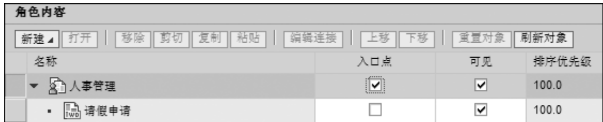


图 6-30　角色配置

3）在弹出的下拉菜单中选择“文件夹”，如图 6-31 所示。

4）在弹出的对话框中输入文件夹名称“考勤”作为二级菜单，随后单击“确定”按钮，如图 6-32 所示。



图 6-31　创建文件夹菜单选择



图 6-32　输入文件夹名称

5）完成创建后，可以在角色内容中看到实际效果，如图 6-33 所示。

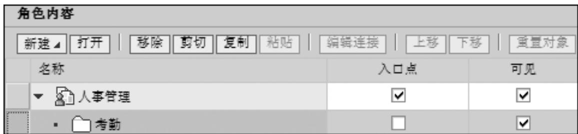


图 6-33　角色配置

6）选中新建的文件夹，并在门户目录中找到需要添加的页面，通过“增量链接”的方式添加到“考勤”目录下，如图 6-34 所示。

名称	入口点	可见
▼ 人事管理	☑	☑
▼ 考勤	☐	☑
▪ 考勤录入	☐	☑

图 6-34　角色配置

7）角色内容的右侧仍然可以看到导航结构的预览，如图 6-35 所示。

8）接着再次使用测试用户 test01 进行登录后，就能看到更新后的三级目录了，如图 6-36 所示。



图 6-35　导航结构预览　　　　图 6-36　运行效果

2. 工作集配置

1）返回最开始创建的自定义文件夹“项目内容”，右击文件夹，如图 6-37 所示。

2）在弹出的菜单中选择“新建”→“工作集”，如图 6-38 所示。

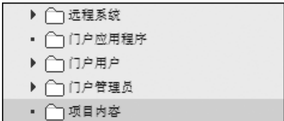


图 6-37　门户内容目录



图 6-38　操作菜单

3）在“工作集名称”和“工作集标识”文本框中分别输入“考勤”和“Atten”，如图 6-39 所示。

4）接着选择新建的工作集，右击选择“打开工作集”，右侧将显示工作集内容，可以看到“工作集内容”和“角色内容”的菜单，如图 6-40 所示。

步骤1:
工作集名称: *
考勤
工作集标识:
Atten
工作集标识前缀（例如：com.公司名称）:
com.myproject
主语言:
中文

图 6-39　输入文件夹信息

工作集内容

新建 | 打开 | 移除 | 剪切 | 复制 | 粘贴 | 上移 | 下移 | 重置对象 | 刷新对象

名称	入口点	可见
▪ 考勤	☐	☑

图 6-40　工作集配置

5）接着将页面添加至工作集，在打开工作集的前提下，右击需要添加的页面，选择“将页面添加至工作集”→“增量链接”，如图 6–41 所示。

6）完成页面添加后，工作集中就能看到完成后的目录结构，如图 6–42 所示。

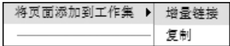


图 6–41　操作菜单

工作集内容

新建　打开　移除　剪切　复制　粘贴　上移　下移　重置对象　刷新对象

名称	入口点	可见
考勤	☐	☑
考勤录入	☐	☑

图 6–42　工作集配置

7）打开“人事管理”角色，再选中工作集，右击并选择“将工作集添加到角色”→“增量链接”，如图 6–43 所示。

8）在“角色内容”中就能看到工作集添加完成后的效果，同时工作集下面的子对象也会同样添加至角色中，如图 6–44 所示。

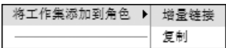


图 6–43　操作菜单

角色内容

新建　打开　移除　剪切　复制　粘贴　编辑连接　上移　下移　重置对象　刷新对象

名称	入口点	可见	排序优先级	合并
人事管理	☑	☑	100.0	是*
考勤	☐	☑	100.0	是*
考勤录入	☐	☑	100.0	是*
考勤	☐	☑	100.0	是*
考勤录入	☐	☑	100.0	是*

图 6–44　角色配置

9）随后使用测试用户登录门户，就能看到完成两种配置后的门户页面，如图 6–45 所示。



图 6–45　运行效果

10）取消选中文件夹目录中的“可见”复选框，如图 6–46 所示。

角色内容

新建　打开　移除　剪切　复制　粘贴　编辑连接　上移　下移　重置对象　刷新对象

名称	入口点	可见	排序优先级
人事管理	☑	☑	100.0
考勤	☐	☐	100.0
考勤录入	☐	☑	100.0
考勤	☐	☑	100.0
考勤录入	☐	☑	100.0

图 6–46　角色配置

11）导航结构预览中会出现修改后的菜单概览，如图 6-47 所示。

12）随后使用测试用户登录门户，就能看到其中一个二级菜单已被隐藏，如图 6-48 所示。



图 6-47　导航结构预览　　　　图 6-48　运行效果

3. 角色间配置

除了以上两种方式以外，其实角色之间也可以互相嵌套，被添加的角色作为另一个角色的子结点，而实际门户中的显示菜单则同样通过“入口点”的设置来决定。

1）首先在文件夹中新建一个“考勤”的门户角色，右击文件夹进行创建，如图 6-49 所示。

2）随后打开角色，并以增量链接的形式添加需要展示的页面，如图 6-50 所示。

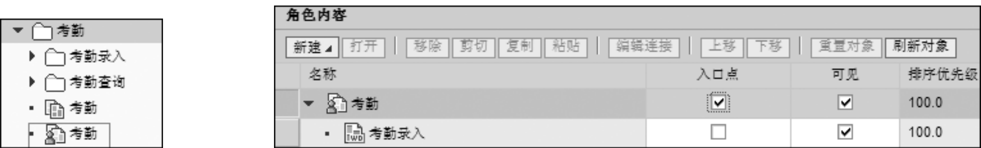


图 6-49　文件夹内容　　　　图 6-50　角色配置

3）接着，先选中并打开分配给用户的“人事管理”角色，再选择将要添加的“考勤”角色（见图 6-51），右击“考勤”并选择“将角色添加到角色”→“增量链接”。

4）仍保持“人事管理”的入口点为选中状态，在右侧的导航结构预览可以看到对应的菜单结构，如图 6-52 所示。由于第一个文件夹类型的“考勤”为不可见，因此预览里只有两个“考勤”目录。



图 6-51　角色配置　　　　图 6-52　导航结构浏览

5）使用测试用户登录企业门户后，就能看到实际的菜单效果了，如图 6-53 所示。



图 6-53　运行效果

6.2.5　多样化的视图模板

企业门户提供了非常多的模板用于创建门户视图，部分类型只需要通过门户中的一些向导配置，而一些涉及后台系统数据处理、逻辑交互的页面，则需要通过 Web Dynpro 进行二次开发才能在门户中完成无缝集成，如图 6-54 所示。

下面以较为简单的 URL 类型的门户视图创建为例，对配置过程进行介绍。

开发实例：

1）首先在自定义文件夹中，创建一个新文件夹“iView”（参考之前的创建过程，右击“创建”），用于管理接下来创建的测试视图，如图 6-55 所示。

2）接着选择新建的“iView”文件夹并右击，在弹出的快捷菜单中选择“新建”→“iView”→“来自模板的 iView”，如图 6-56 所示。



图 6-54　不同类型的 iView



图 6-55　创建后的目录结构

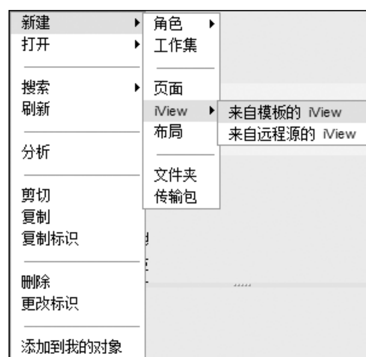


图 6-56　操作菜单

3）在右侧新建 iView 的向导中选中“URL iView”单选按钮，随后单击“下一步”按钮，如图 6-57 所示。

4）输入新建视图的名称、标识和标识前缀，随后单击“下一步”按钮，如图 6-58 所示。

5）由于选择的是互联网链接类型的视图，因此需要输入对应的 URL 链接，同时右侧也有按钮可以进行网页的预览，用于确认输入的地址是否正确。本实例使用之前已发布至服务器端的 dynp08 工程应用程序的链接，如图 6-59 所示。

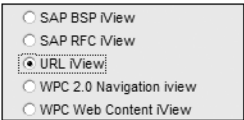


图 6-57　选择类型

名称: *
请假申请
iView 标识:
iView 标识前缀（例如: com.公司名称):
com.myproject
主语言:
英文

图 6-58　输入对象信息

6）在完成门户视图的创建后，在相应的目录中就能看到已创建的对象，如图 6-60 所示。

7）接着将视图添加至工作集中，首先右击工作集并打开，随后右击视图，并选择“将 iView 添加到工作集”→“增量链接”，如图 6-61 所示。

输入 URL: *
http://<IP>:50100/webdynpro/resources/demo.sap.com/mydynp08/ReqApp

图 6-59　输入对应链接



图 6-60　创建后的对象

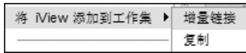


图 6-61　操作菜单

8）右侧工作集内容中就会出现添加完成后的效果，如图 6-62 所示。

请假
请假申请
请假申请

图 6-62　工作集配置

9）由于之前工作集与角色也是增量链接的关系，因此打开角色查看时会发现，角色中也已同步添加了新建的门户视图，如图 6-63 所示。

10）使用测试用户登录企业门户后，就能看到新增的菜单以及测试页面，如图 6-64 所示。

人事管理
考勤
请假
请假申请
请假申请

图 6-63　工作集配置

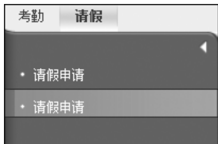


图 6-64　运行效果

6.3 用户管理

SAP 企业门户中的用户管理是通过系统的用户管理引擎（User Management Engine，UME）进行运转和管理的，而使用管理员账号登录后看到的“用户管理”页面只是 UME 的一小部分。UME 是由连接至底层 SAP Java 应用服务器以及管理所有用户和组原始数据的存储的一系列用户管理服务组成的，所以门户中的应用程序不需要单独访问用户数据和服务，而是通过 UME 进行集中的管理，如图 6-65 所示。

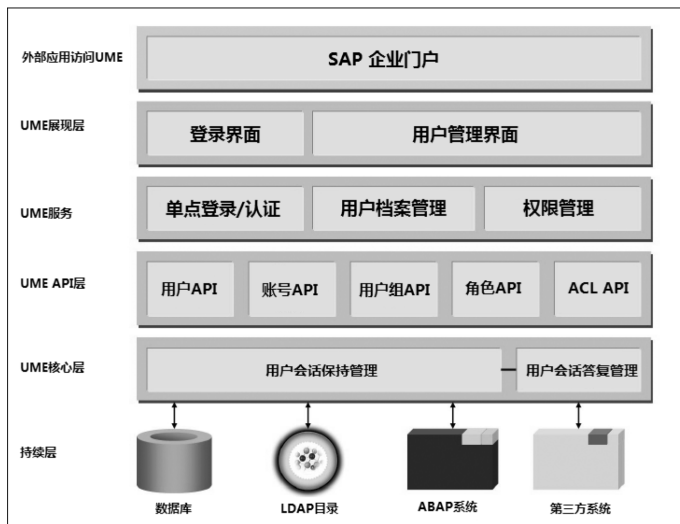


图 6-65 UME 整体架构示意图

虽然整个 UME 的架构看起来有点吓人，但在实际配置中，大部分的工作都是在“UME 的展现层”中的“用户管理”界面完成的。用户管理一共包含了以下 3 个对象：

- User（用户）：对应企业门户账号的实际用户，可维护基本信息、包含角色、分配的组以及与其他系统的用户映射。
- Role（角色）：在内容管理中进行定义，在用户管理中进行分配，可分配给用户或者组。
- Group（组）：在用户管理中进行定义，可以与角色和用户进行双向的分配。

用户可以分配多个角色，也可以分配多个组，角色的组合可以归集到一个组，同时角色与活动或操作关联，来间接地实现门户中用户角色或组的不同显示不同的页面活动与操作。当然这里的角色也不仅仅限于门户角色，所有的后台角色和后台组都遵循这个关系。三者之间的组合也十分灵活。

6.3.1 账号管理

进入 SAP 企业门户的用户管理页面，首先看到的是一个搜索页面，可以输入关键字并选择对应的搜索内容（包括用户、角色和组）进行查询，也支持带“*”的模糊查询。在搜索结果过多时，默认只会显示 200 条记录。

1. 创建和修改

首先介绍如何创建一个用户，以及用户信息的修改。

开发实例：

1）进入“用户管理”页面，单击“创建用户”按钮，如图 6-66 所示。注意，根据搜索条件的不同，创建按钮会同时发生变化。



图 6-66　搜索用户界面

2）接着下方会出现填写用户详细信息的页面，输入“登录标识”“定义密码”“确认密码”和“姓”即可完成最基本的一个用户创建，如需要添加组或者角色，则分别在“分配的组”和“分配的角色”页签中进行配置，如图 6-67 所示。

详细信息
修改 | 保存 取消
常规信息 | 账户信息 | 联系信息 | 附加信息 | 分配的角色 | 分配的组
登录标识: * test01
定义初始密码
生成密码
禁用密码
定义密码: * ••••••
确认密码: * ••••••
姓: * test01
名:
电子邮件地址:
称谓:
语言:
激活可存取性功能:
安全策略: 缺省
唯一标识:

图 6-67　输入用户信息

3）同时，也可以对该用户定义具体的账户信息，如图 6-68 所示。

详细信息
修改 | 保存 取消
常规信息 | 账户信息 | 联系信息 | 附加信息 | 分配的角色 | 分配的组
已锁定的用户账户: 选择以设置管理锁。取消选择以解除管理和密码锁
账户有效起始日期:
账户有效结束日期:
登录别名:

图 6-68　维护账户信息界面

4）定义具体的联系信息，如图 6-69 所示。

5）由于初始密码是管理员在创建用户时进行设置的，因此新用户默认在首次登录时，

需要修改密码。在首次登录修改密码后，就可以正常地登录和使用了，如图 6-70 所示。

图 6-69　维护联系信息界面

图 6-70　用户登录时修改密码界面

2. 解锁与锁定

用户账号有“可用”和“锁定”两种状态，在锁定状态下，账户将无法登录。在尝试登录失败次数过多时，或者管理员手动进行账号锁定，都会使得账号处于“锁定”状态。

1）当测试账号 test01 因为密码输入错误次数过多而锁定时，使用管理员账号登录企业门户，搜索“test01”用户，在查询结果中可以看到，账户的状态是“锁定”状态，图标显示为灰色，如图 6-71 所示。

2）如需要解锁，首先在表格中选中该条目（该用户所在行），接着单击“解锁”按钮，如图 6-72 所示。

图 6-71　已锁定用户

图 6-72　解锁对应用户

3）会弹出一个提示解锁的原因的文本框，可以输入一些提醒消息给用户，单击“解锁”按钮即可，如图 6-73 所示。

4）解锁后，用户状态恢复正常，如图 6-74 所示。

图 6-73　输入解锁原因

图 6-74　解锁后的用户状态

5）如果用户忘记了登录密码，解锁账户并不能帮助用户解决问题，则需要进行重置密码的操作。还是选中需要修改密码的用户，单击“修改”按钮，随后单击“账户”信息的页签，输入“定义密码”和“确认密码”，单击“保存”按钮即可，如图 6-75 所示。

6）由于企业门户的安全设置，如果账号不是技术类型用户（一般用户均不建议使用该用户类型），则在登录时会提醒首次登录需要修改密码，如图 6-76 所示。

图 6-75　修改用户信息界面

图 6-76　用户登录时修改密码界面

7）UME 默认的用户安全策略为“默认”与“技术用户”。普通的前台用户创建时安全策略为默认，默认的安全策略设置（如密码管理、登录标识限制等）都会对前台用户生效。由于系统间通信的用户验证往往不需要频繁地去做密码变更，因此这时就需要创建技术用户作为后台配置的用户验证。这个属性也沿袭了 SAP 在 ERP 系统中的用户类型。具体在修改用户信息时，可以在基本信息中看到“安全策略”以及对应的可选类型，如图 6-77 所示。

3. 用户组配置

用户角色的分配在 6.2.3 节已进行过介绍，接着通过一个实例介绍用户组的创建和配置。

1）首先创建一个测试用的用户组，将“搜索条件”中的搜索类型改为“组”（默认是“用户”），随后单击“创建组”按钮，如图 6-78 所示。

图 6-77　用户信息的安全策略

图 6-78　创建组的操作页面

2）在表格下方出现用户组的具体信息，输入唯一名称、描述后保存即可，如图 6-79 所示。如果需要分配角色，则进入“已分配的角色”页签，参考用户的角色的分配进行添加即可。

图 6-79　输入组名称

3）接着再测试一下将用户组分配给角色首先将搜索条件改为“角色”，单击“创建角色”按钮，如图 6-80 所示。

4）输入角色的唯一名称和描述，随后进入“分配的组”页签，如图 6-81 所示。

图 6-80　创建角色的操作页面

图 6-81　输入角色信息

5）在左侧可以搜索用户管理中定义的用户组，添加后会出现在右侧，如图 6-82 所示。



图 6-82　修改分组的界面

6）添加完成后，单击“保存”按钮即完成所有配置，如图 6-83 所示。



图 6-83　查看已分配的组

6.3.2　批量管理

在实施的项目场景中，尤其是项目上线时，管理员不可能像之前的配置页面一样，一个一个地手动去创建用户，这时就需要使用脚本进行用户的批量导入。

开发实例：

1）首先单击“输入”，进入批量导入的页面，如图 6-84 所示。



图 6-84　导入数据界面

2）导入数据支持两种方式：一种是本地文件上传，一种是直接录入脚本的文本内容进行导入，如图 6-85 所示。参考导入脚本如下：

```
[User]
uid = test01
last_name = 01
first_name = test
accessibility = 0
role = pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.PR_HRMNG;
[User]
uid = test02
last_name = 02
first_name = test
accessibility = 0
role = pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.PR_HRMNG;
```

纯文本上传:

```
[User]
uid=test01
last_name=01
first_name=test
accessibility=0
role=pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.PR_HRMNG;
[User]
uid=test02
last_name=02
first_name=test
accessibility=0
role=pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.PR_HRMNG;
```

图 6–85　脚本参考

注：脚本文件为 TXT 格式，文件内容与录入脚本需要遵守同样的编码规范。

3）在脚本录入之后或选择本地脚本文件后，选中“覆盖现有数据”复选框，单击“上载”按钮，如图 6–86 所示。



图 6–86　上载数据界面

4）在页面中会返回所有导入结果，对于系统已存在的用户将进行覆盖，而对于不存在的用户则会自动根据脚本中的信息进行创建，如图 6–87 所示。

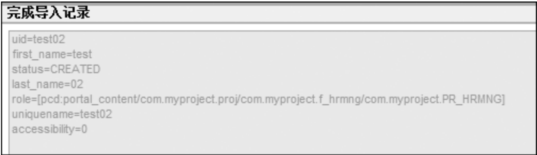
全部（2）　返回到导入

状态	显示名称	唯一名称
已更新	test01	test01
已创建	test02	test02

图 6–87　导入数据结果

5）单击任意一个用户可以查看完整的导入记录，如图 6–88 所示。

6）进入用户查询界面也可以看到对应用户已被创建，如图 6–89 所示。



完成导入记录

uid=test02
first_name=test
status=CREATED
last_name=02
role=[pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.PR_HRMNG]
uniquename=test02
accessibility=0

图 6–88　详细的导入记录



图 6–89　查看已创建的用户

注：脚本导入这个方法可以用于批量的用户创建，也可以用于角色创建和用户组创建；还可以对用户进行批量的密码初始化和角色分配，但在进行脚本操作前，建议对现有用户进行备份。

7）如果需要对现有用户进行备份，在用户搜索的界面，选中需要备份的用户，随后单击“导出”按钮即可，如图 6–90 所示。

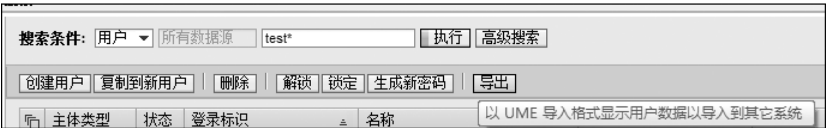


图 6–90　导出用户信息的操作界面

8）随后会出现对应用户的脚本信息，不过并不会包含用户的密码信息，如图 6-91 所示。

6.3.3　用户管理 API 开发

SAP 提供了一系列用户管理的 API，通过 UME 的 API 可以完成绝大部分的用户管理功能，包括新建组、角色或用户等，同时也可以实现对组添加角色，给指定用户分配角色等操作。对于较为复杂、特殊的用户需求，可以考虑通过 API 自定义用户管理，这样会更加灵活。

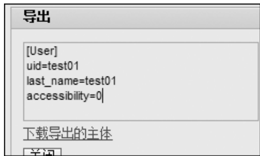


图 6-91　导出的脚本内容

1. 账号管理

1）新建一个用于测试的 Web Dynpro 项目，在 DI 透视图中手动添加对“tc/je/usermanagement/api”开发组件的依赖，如图 6-92 所示。

2）在 Web Dynpro 视图中添加对应的“新增用户”按钮，如图 6-93 所示。

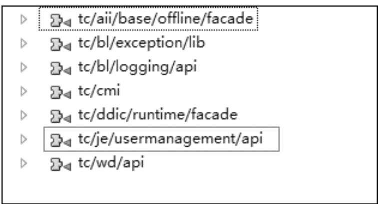


图 6-92　需要依赖的开发组件清单



图 6-93　运行效果

3）在 Java 编辑器中添加按钮事件的处理逻辑：

```
final IUserFactory userFactory = UMFactory. getUserFactory( ) ;
final IUserAccountFactory userAccountFactory = UMFactory
        . getUserAccountFactory( ) ;
try {
// create UserMaint 包含用户个人信息
String accountNo = "test04" ;                    //账号
IUserMaint userMaint = userFactory. newUser( accountNo) ;
userMaint. setDisplayName( "test04" ) ;
userMaint. setFirstName( "04" ) ;                //名
userMaint. setLastName( "test" ) ;               //姓
userMaint. setPersonID( "0004" ) ;               //工号
userMaint. setCompany( "SAPEYES" ) ;             //公司
userMaint. setEmail( "test04@ sapeyes. com" ) ;  //邮箱
// create UserAccount 包含账号认证信息
IUserAccount userAccount = userAccountFactory
. newUserAccount( accountNo) ;
userAccount. setLocked( false, 0) ;              //默认不锁定
userAccount. setSecurityPolicy( IUserAccount. SECURITY_POLICY_TYPE_TECHNICAL) ;
                                                 //用户安全策略为技术用户
userAccount. setPassword( "1234qwer" ) ;         //设置技术用户的密码
// Write the changes to the user store 提交用户的创建
userFactory. commitUser( userMaint, userAccount) ;
} catch (UMException e) {
logger. errorT( "新增 UME 用户数据失败:" + e. getMessage( ) ) ;
}
```

4）在视图中再添加一个“删除用户”的按钮，如图 6-94 所示。

5）在 Java 编辑器中添加按钮事件的处理逻辑：

```
final IUserFactory userFactory = UMFactory. getUserFactory( ) ;
final IUserAccountFactory userAccountFactory = UMFactory
. getUserAccountFactory( ) ;
try {
String accountNo = "test04" ;                    //账号
userFactory. deleteUser( userFactory. getUserByLogonID( accountNo)
. getUniqueID( ) ) ;                             //删除账号
} catch (UMException e) {
logger. errorT( "删除 UME 用户数据失败:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
"删除 UME 用户数据失败:" + e. getMessage( ) ) ;
}
```

6）在视图中再添加一个“查询用户”按钮，如图 6-95 所示。



图 6-94　运行效果 1

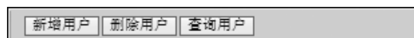


图 6-95　运行效果 2

7）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IUserFactory userFact = UMFactory. getUserFactory( ) ;
IUser user;
try {
user = userFact. getUserByLogonID( "test04" ) ;
if (user !  = null) {
String approver = user. getUniqueID( ) ;
wdComponentAPI. getMessageManager( ). reportSuccess(
"UniqueId:" + approver) ;
} else {
wdComponentAPI. getMessageManager( ). reportException( "不存在的用户!" ) ;
}
} catch (Exception e) {
logger. errorT( "UME 读取用户数据异常:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
"UME 读取用户数据异常:" + e. getMessage( ) ) ;
}
```

8）完成开发后，保存所有修改，编译并部署应用程序即可进行功能测试，如图 6-96 所示。

9）单击“查询用户”按钮，读取用户失败并抛异常，如图 6-97 所示。

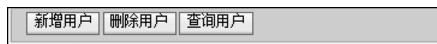


图 6-96　运行效果 3

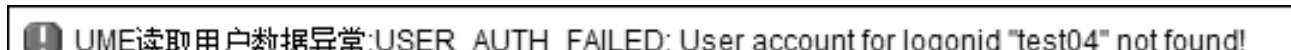


图 6-97　提醒消息

10）单击“新增用户”按钮后，再次单击“查询用户”按钮，打印新增用户的唯一标识，表示创建用户成功，如图 6-98 所示。

11）使用管理员登录系统后，可以在 UME 用户信息中看到具体的用户，如图 6-99 所示。

UniqueId:USER.PRIVATE_DATASOURCE.un:test04

图 6-98　成功创建的消息

登录标识:	test04
姓氏:	test
名字:	04
电子邮件地址:	test04@sapeyes.com
称谓:	
语言:	
激活可存取性功能:	☐
安全策略:	技术用户
唯一标识:	USER.PRIVATE_DATASOURCE.un:test04

图 6-99　用户详细信息

12）单击“删除用户”按钮后，再次单击“查询用户”按钮，再次抛出异常，标识删除成功，如图 6-100 所示。

2. 角色管理

1）继续使用之前的 Web Dynpro 项目，在视图中添加一个“新增角色”按钮，如图 6-101 所示。

UME读取用户数据异常:USER_AUTH_FAILED: User account for logonid "test04" not found!

图 6-100　提醒消息

新增用户 | 删除用户 | 查询用户 | 新增角色

图 6-101　运行效果 4

2）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IRoleFactory roleFactory = UMFactory. getRoleFactory( ) ;
try {
IRole newRole = roleFactory. newRole( "ROLE02" ) ;
newRole. setDisplayName( "ROLE02" ) ;
newRole. commit( ) ;
} catch (Exception e) {
logger. errorT( "UME 新增角色数据异常:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
"UME 新增角色数据异常:" + e. getMessage( ) ) ;
}
```

3）在视图中添加一个“删除角色”按钮，如图 6-102 所示。

4）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IRoleFactory roleFactory = UMFactory. getRoleFactory( ) ;
try {
roleFactory. deleteRole( roleFactory. getRoleByUniqueName( "ROLE02" )
. getUniqueID( ) ) ;
} catch (Exception e) {
logger. errorT( "UME 删除角色数据异常:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
"UME 删除角色数据异常:" + e. getMessage( ) ) ;
}
```

5）在视图中添加一个“查询角色”按钮，如图 6-103 所示。

新增用户 | 删除用户 | 查询用户 | 新增角色 | 删除角色

图 6-102　运行效果 5

新增用户 | 删除用户 | 查询用户 | 新增角色 | 删除角色 | 查询角色

图 6-103　运行效果 6

6）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IRoleFactory roleFactory = UMFactory. getRoleFactory( ) ;
IUser user;
try {
IRole roleByUniqueName = roleFactory. getRoleByUniqueName( " ROLE02" ) ;
if ( roleByUniqueName ！ = null) {
String role = roleByUniqueName. getUniqueID( ) ;
wdComponentAPI. getMessageManager( ). reportSuccess(
" UniqueId:" + role) ;
} else {
wdComponentAPI. getMessageManager( ). reportException( " 不存在的角色!" ) ;
}
} catch (Exception e) {
logger. errorT( " UME 读取角色数据异常:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
" UME 读取角色数据异常:" + e. getMessage( ) ) ;
}
```

7）在视图中添加一个“添加用户到角色”按钮，如图 6-104 所示。

8）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IRoleFactory roleFactory = UMFactory. getRoleFactory( ) ;
try {
//根据用户的唯一 ID 添加用户到角色 roleFactory. addUserToRole( UMFactory. getUserFactory( )
. getUserByLogonID( " test04" )
. getUniqueID( ) , roleFactory. getRoleByUniqueName( " ROLE02" ). getUniqueID( ) ) ;
} catch (Exception e) {
logger. errorT( " UME 添加用户到角色异常:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
" UME 添加用户到角色异常:" + e. getMessage( ) ) ;
}
```

9）完成开发后，保存所有修改，编译并部署应用程序即可进行功能测试，如图 6-105 所示。

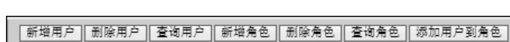


图 6-104　运行效果 7

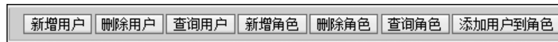


图 6-105　运行效果 8

10）单击“查询角色”按钮，读取角色失败并抛出异常信息，如图 6-106 所示。

11）单击“新增角色”按钮后，再次单击“查询角色”按钮，打印新增角色的唯一标识，表示创建角色成功，如图 6-107 所示。

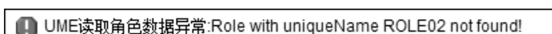


图 6-106　提示消息 1

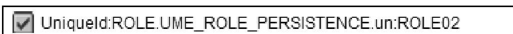


图 6-107　提示消息 2

12）使用管理员登录系统后，可以在 UME 用户信息中看到具体的用户，如图 6-108 所示。

13）单击“删除角色”按钮后，再次单击“查询角色”按钮，再次抛出异常，表示删除成功，如图 6-109 所示。

唯一名称: ROLE02
显示名称: ROLE02
位置:
描述:
唯一标识: ROLE.UME_ROLE_PERSISTENCE.un:ROLE02

图 6-108　角色的详细信息

UME读取角色数据异常:Role with uniqueName ROLE02 not found!

图 6-109　提示消息

14）单击“添加用户到角色”按钮，随后使用管理员登录系统后，可以在 UME 角色信息中看到已分配的用户，如图 6-110 所示。

已分配的用户

搜索条件: 所有数据源 　 递归搜索 执行

主体类型	登录标识	名称
	test04	test04

图 6-110　已分配的用户

3. 用户组管理

1）继续使用之前的 Web Dynpro 项目，在视图中添加一个“新增组”按钮，如图 6-111 所示。

新增用户 删除用户 查询用户 新增角色 删除角色 查询角色 添加用户到角色 新增组

图 6-111　运行效果 9

2）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IGroupFactory groupFactory = UMFactory. getGroupFactory( ) ;
try {
IGroup newGroup = groupFactory. newGroup( "GROUP02" ) ;
newGroup. setDisplayName( "ROLE02" ) ;
newGroup. commit( ) ;
} catch (Exception e) {
logger. errorT( "UME 新增组数据异常:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
"UME 新增组数据异常:" + e. getMessage( ) ) ;
}
```

3）在视图中添加一个“删除组”按钮，如图 6-112 所示。

新增用户 删除用户 查询用户 新增角色 删除角色 查询角色 添加用户到角色 新增组 删除组

图 6-112　运行效果 10

4）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IGroupFactory groupFactory = UMFactory. getGroupFactory( ) ;
try {
groupFactory. deleteGroup( groupFactory. getGroupByUniqueName(
"GROUP02" ). getUniqueID( ) ) ;
} catch (Exception e) {
```

```
logger.errorT("UME 删除组数据异常:" + e.getMessage());
wdComponentAPI.getMessageManager().reportException(
"UME 删除组数据异常:" + e.getMessage());
}
```

5）在视图中添加一个“查询组”按钮，如图 6-113 所示。



图 6-113　运行效果 11

6）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IGroupFactory groupFactory = UMFactory.getGroupFactory();
try {
IGroup groupByUniqueName = groupFactory.getGroupByUniqueName("GROUP02");
if (groupByUniqueName != null) {
String group = groupByUniqueName.getUniqueID();
wdComponentAPI.getMessageManager().reportSuccess(
"UniqueId:" + group);
} else {
wdComponentAPI.getMessageManager().reportException("不存在的组!");
}
} catch (Exception e) {
logger.errorT("UME 读取组数据异常:" + e.getMessage());
wdComponentAPI.getMessageManager().reportException(
"UME 读取组数据异常:" + e.getMessage());
}
```

7）在视图中添加一个“添加角色到组”按钮，如图 6-114 所示。

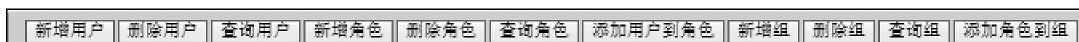


图 6-114　运行效果 12

8）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IRoleFactory roleFactory = UMFactory.getRoleFactory();
try {
//通过角色的 UniqueId 添加角色到组
roleFactory.addGroupToRole(UMFactory.getGroupFactory()
.getGroupByUniqueName("GROUP02").getUniqueID(), UMFactory
.getRoleFactory().getRoleByUniqueName("ROLE02")
.getUniqueID());
} catch (Exception e) {
logger.errorT("UME 添加角色到组异常:" + e.getMessage());
wdComponentAPI.getMessageManager().reportException(
"UME 添加角色到组异常:" + e.getMessage());
}
```

9）在视图中添加一个“添加用户到组”按钮，如图 6-115 所示。



图 6-115　运行效果 13

10）在 Java 编辑器中添加按钮事件的处理逻辑：

```
IGroupFactory groupFactory = UMFactory. getGroupFactory( ) ;
try {
//通过用户的 UniqueId 添加用户到组
groupFactory. addUserToGroup( UMFactory. getUserFactory( )
. getUserByUniqueName( "test04" ). getUniqueID( ) , UMFactory
. getGroupFactory( ). getGroupByUniqueName( "GROUP02" )
. getUniqueID( ) ) ;
} catch (Exception e) {
logger. errorT( "UME 添加用户到组异常:" + e. getMessage( ) ) ;
wdComponentAPI. getMessageManager( ). reportException(
"UME 添加用户到组异常:" + e. getMessage( ) ) ;
}
```

11）完成开发后，保存所有修改，编译并部署应用程序即可进行功能测试。

12）单击“查询组”按钮，读取组失败并抛出异常，如图 6-116 所示。

13）单击“新增组”按钮后，再次单击“查询组”按钮，打印新增组的唯一标识，标识创建组成功，如图 6-117 所示。

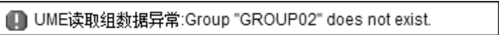


图 6-116　提示消息 1



图 6-117　提示消息 2

14）使用管理员登录系统后，可以在 UME 用户信息中看到具体的组信息，如图 6-118 所示。

15）单击“添加角色到组”按钮，随后使用管理员登录系统后，可以在 UME 角色信息中看到已分配的角色，如图 6-119 所示。

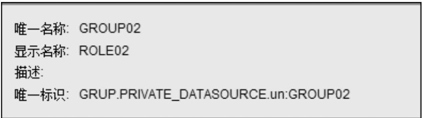


图 6-118　组的详细信息

已分配的角色

搜索条件: 所有数据源 □递归搜索 执行

主体类型	名称	描述
	ROLE02	

图 6-119　查询已分配的角色

16）单击“添加用户到组”按钮，使用管理员登录系统后，可以在 UME 角色信息中看到已分配的用户“test04”，如图 6-120 所示。

已分配的用户

搜索条件: 所有数据源 □递归搜索 执行

主体类型	登录标识	名称
	test04	test04

图 6-120　查看已分配的用户

6.4 系统管理

SAP 企业门户的系统管理是针对门户系统进行参数、服务配置，包括系统配置、传送、监控、许可、系统前景、联合门户、支持和内容统计等内容。门户中的系统管理与 SAP NetWeaver 的 NWA 管理者平台是有区别的，前者是针对门户系统的系统配置，而后者更多的是为 NetWeaver 平台服务，也就是服务器端的配置。从某种意义来说，门户系统其实也只是 NetWeaver 平台上的一个复杂应用，如图 6-121 所示。



图 6-121　UME 配置界面

由于篇幅有限，无法在章节中对系统配置的各个功能点进行说明，本节重点介绍与 Web Dynpro 开发应用相关的一些配置点。

6.4.1 UME 配置

UME 配置中包含了数据源、安全策略、通知电子邮件、用户映射、用户管理 UI 等内容，覆盖了从用户登录、后台用户数据源配置到第三方系统映射等功能，如图 6-122 所示。



图 6-122　UME 配置的详细界面

1. 数据源配置

配置修改 UME 的数据源，可以修改 UME 数据源的连接参数和连接类型，即修改 UME 后台数据库对象。默认情况下，UME 的数据源仅使用 NetWeaver 环境安装时指定的数据库，即在视图中“数据源”属性会显示“仅数据库”，但由于企业的信息系统往往会涵盖较多不同语言、不同架构的子系统，并且系统间采用统一的用户管理，使用相同的用户主数据，如果每个子系统都建立一套独立的用户存储空间，会带来数据同步、数据访问不一致等诸多弊端，因此为了减少因多数据存储带来的系统间用户主数据不一致或用户同步所带来的工作

量，项目中会通过修改前台 EP 的用户数据源，直接实现统一用户数据源的目的。

数据源配置实例：

1）单击视图中的“修改配置”按钮，就可以通过下列列表对 UME 支持的数据源类型进行选择，同时下方多了一个数据源配置文件的“上载控件”按钮，如图 6–123 所示。



图 6–123　修改数据源

2）单击“数据源”右侧的下拉按钮，就能看到所有支持的类型，包括很多知名企业的 LDAP 产品，以及 SAP 的 ABAP 系统，如图 6–124 所示。

3）接着选择 ABAP 系统连接测试环境进行演示，如图 6–125 所示。

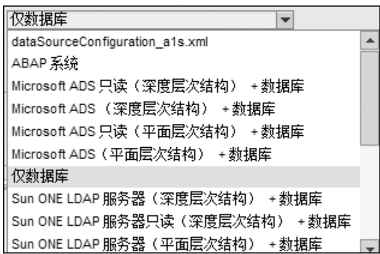


图 6–124　数据源支持的不同类型

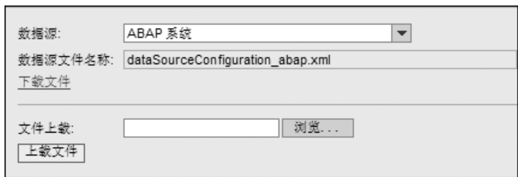


图 6–125　修改为 ABAP 系统数据源

其中“数据源文件名称”会自动切换对应的配置文件，如果需要进行自定义，则可以通过单击“上载文件”进行替换。

4）在选择“ABAP 系统”后，上方配置页签也会多出一栏“ABAP 系统”，单击该页签，创建并维护 RFC 目标，目标指定 ABAP 系统的连接信息，确认配置连接成功，如图 6–126 所示。

这里的 RFC 目标其实就是在 Web Dynpro 开发中使用 RFC 模型时配置的逻辑目标，同样地用于两个系统之间的连接。完成配置后，单击“保存”按钮，并重启服务器实例。

注： *修改数据源需要慎重，建议在备份必要用户数据后进行。*

5）接着进入用户管理页面，查询 ABAP 系统中的用户，如图 6–127 所示。

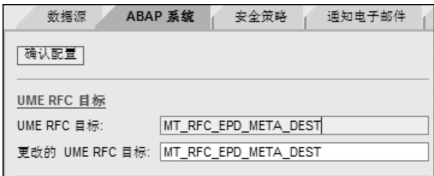


图 6–126　ABAP 系统目标配置

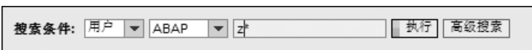


图 6–127　搜索用户界面

6）可以看到搜索结果的用户清单中，“数据源”一列显示的是“ABAP”，也就表示这些用户来自于 ABAP 系统，如图 6-128 所示。

部门	数据源
开发	ABAP
开发	ABAP
开发	ABAP
开发	ABAP

图 6-128　查询用户清单的数据源

注：切换数据源只是省略了创建用户的动作，企业门户中的角色分配仍然需要具体配置。

2. 其他参数配置

1）安全策略：用户登录的安全验证方式，包括默认与技术用户，也可以自定义新的安全策略。UME 中新建用户的属性中也包含安全策略。可以修改默认的安全策略，如登录标识长度、密码长度等，如图 6-129 所示。

2）登录标识相关的安全策略设置，如图 6-130 所示。

数据源　安全策略　通知电子邮件　用户

安全策略参数文件

名称
默认
技术用户

创建　删除

图 6-129　安全策略配置

与登录标识相关的属性

登录标识的最小长度:	1
登录标识的最大长度:	20
登录标识中数字的最小数:	0
在登录标识中的特殊字符最小数:	0
登录标识中小写字母的最小数目:	0

图 6-130　登录标识相关配置

3）密码相关的安全策略设置，如图 6-131 所示。

4）锁定相关的安全策略设置，如图 6-132 所示。

与密码相关的属性

密码的最小长度:	5
密码的最大长度:	14
密码中混合写字母的最小数:	0
密码中字母数字字符的最小数:	1
密码中特殊字符的最小数:	0
密码历史记录的大小:	0
不允许的口令:	
密码最长空闲时间:	0
密码有效性期间（天数）:	90

☐ 允许登录标识作为部分密码
☐ 允许旧密码作为部分新密码
☑ 允许用户更改自有密码
☐ 增强登录时的密码案例策略

图 6-131　密码相关的安全策略设置

与用户账户锁定相关的属性

失败登录尝试的最大次数:	6
自动解锁时间（分钟数）:	60

图 6-132　锁定相关的安全策略设置

通过修改这些参数，改变登录标识、密码修改复杂度、用户过期时间等。技术用户是针对系统间通信而设置的后台验证策略，用户的安全策略为技术用户时，不需要修改初始密码，而且密码没有过期时间。

5）用户管理 UI：控制用户管理界面的搜索结果显示，如果允许的搜索结果过大，则会影响服务器在用户管理搜索时的效率，如图 6-133 所示。

6）登录帮助，以及是否需要开启安全问题策略，如图 6-134 所示。

搜索显示并显示表
搜索结果的最大数: 1000
大搜索结果的警告临界值: 200
在大显示表中的行数: 20
中等显示表中的行数: 10
在小显示表中的行数: 5

图 6-133　搜索界面的配置

Logon Help
☐ 登录帮助中需要姓和名
○ 使用预定义的安全问题
○ 启用安全问题
⊙ 禁用安全问题

图 6-134　登录帮助相关配置

6.4.2　内容传输

由于 SAP 企业门户的内容变更并没有 ERP 系统或 Web Dynpro 开发那么频繁，大部分的变更在项目上线前都已完成，因此门户内容在不同系统之间的传输是以传输包的形式进行的，从而减少了生产传输的配置工作量，如图 6-135 所示。

1. 传输包导出

在系统管理的“传送”页面，可以看到与企业门户内容管理同样的菜单，但唯一的区别是，只有在系统管理的“传送”菜单下，才能看到自定义配置的传输包内容。



图 6-135　门户内容传输的菜单

1）首先，创建一个传输包的存放文件夹，右击“门户内容”，选择新建文件夹，然后输入对象名称、对象标识和文件夹 ID Prefix，如图 6-136 所示。

2）完成后，文件夹与内容管理中进行测试的目录“项目内容”并列，如图 6-137 所示。

3）右击“项目内容传输包”，选择“新建”→“传输包”，在文件夹中创建传输包，如图 6-138 所示。

对象名称: *
项目内容传输包
对象标识 （要自动创建标识，则保留此字段为空）:
文件夹 ID Prefix (Example: com.companyname):
com.myproject
主语言
中文

图 6-136　输入对象信息

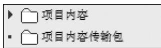


图 6-137　文件夹目录



图 6-138　操作菜单

4）输入对象名称、对象标识和传输包 ID Prefix，完成传送包的创建，如图 6-139 所示。

5）创建成功后，可以在文件夹中看到一个包裹图标的文件，如图 6-140 所示。

6）由于这个传输包仍然是一个空的“箱子”，因此需要添加内容。右击“tspk_001”，并选择“打开”→“包内容”，如图 6-141 所示。

7）在打开传送包后，可以选中需要传输的内容进行添加，这个过程与添加页面到角色中十分相似。右击“项目内容”文件夹，选择“添加到传送包”→“所有对象”。如果这



图 6-139　输入对象信息

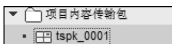


图 6-140　创建后的对象

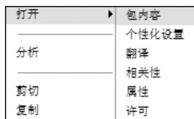


图 6-141　操作菜单

里选择“对象”，则仅会添加选中的文件夹对象，而不会包含所选文件夹中的对象，所以一般都会选择添加“所有对象”，如图 6-142 所示。

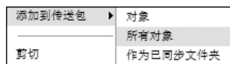


图 6-142　操作菜单

8）添加完成后，在右侧的“传送包内容”中就能看到添加后的目录清单，最右侧的状态表示是否添加成功，如图 6-143 所示。

传送包内容

内容编辑器使您能够查看并管理传送包内容

移除 | 打开 | 预览 | 在门户目录中定位

名称	标识	状态
门户内容	pcd:portal_content	✓
项目内容	pcd:portal_content/com.myproject.proj	✓
IVIEW	pcd:portal_content/com.myproject.proj/com.myproject.IVIEW	✓
请假申请	pcd:portal_content/com.myproject.proj/com.myproject.IVIEW/com.myproject...	✓
人事管理	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng	✓
考勤	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproj...	✓
考勤录入	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproj...	✓
考勤录入	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproj...	✓
考勤查询	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproj...	✓
考勤查询	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproj...	✓

图 6-143　传送包内容清单

9）接着右击传送包，选择“导出”→“快速导出”，如图 6-144 所示。这里如果选择“标准导出”会多一个环节，需要进行导出内容的确认，可以根据实际情况选择，导出的结果是一样。

10）在右侧的导出向导中，可以看到导出传送包的简单信息，确认无误后，单击“导出”按钮，如图 6-145 所示。



图 6-144　操作菜单

上一步 | 导出 | 完成 | 取消

传送明细:

传送方法:	FileSystem
文件名:	com.myproject.tspk_0001_20140213_084315.epa
文件目标:	/usr/sap/MPD/SYS/global/pcd/Export

传送包属性:

导出模式:	所有数据类型
隐含参考:	已包括参考
明确参考:	已包括参考
已过滤的参考:	com.myproject.*

传送包内容:

用于导出的项:	27
排除的项:	0

图 6-145　导出向导

11）在导出过程中，页面会出现导出的进度条，如图 6-146 所示。

12）在导出完成后，页面会出现导出结果的一个日志信息，如图 6-147 所示。

13）也可以看到导出对象的明细清单，如图 6-148 所示。

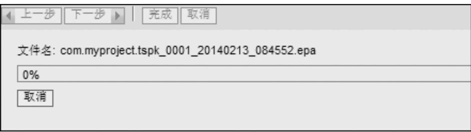


图 6-146　导出进度条

导出汇总

要导出的项目	27
已导出	27
已从包中排除	0
错误	0

图 6-147　导出日志

导出的对象

显示状态: 全部

类型	名称	持续时间（毫秒）	标识	状态
	tspk_0001	55	pcd:portal_content/com.myproject.New_Folder/com.mypro…	确定
	固定资产维护	4	pcd:portal_content/com.myproject.proj/com.myproject.f_lo…	确定
	固定资产维护	6	pcd:portal_content/com.myproject.proj/com.myproject.f_lo…	确定
	固定资产作废	4	pcd:portal_content/com.myproject.proj/com.myproject.f_lo…	确定
	固定资产作废	6	pcd:portal_content/com.myproject.proj/com.myproject.f_lo…	确定
	后勤管理	3	pcd:portal_content/com.myproject.proj/com.myproject.f_lo…	确定
	考勤	10	pcd:portal_content/com.myproject.proj/com.myproject.f_hr…	确定
	考勤	5	pcd:portal_content/com.myproject.proj/com.myproject.f_hr…	确定
	考勤	4	pcd:portal_content/com.myproject.proj/com.myproject.f_hr…	确定

图 6-148　导出对象的明细清单

14）在默认情况下，该导出向导是将生成的“.epa”传送包文件导出至服务器端的“export”目录中，但为了避免频繁地进行服务器登录、文件传输操作，门户中也允许用户将导出文件直接下载至本地计算机中，方便后续的操作，如图 6-149 所示。

2. 传输包导入

1）以管理员身份登录传输包导入的目标系统，打开“系统管理”→“传送”→“导入”的页面，“包文件的资源”选择“客户端”，随后单击“浏览”按钮，选择并上载本地计算机中的传送包文件，如图 6-150 所示。

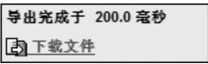


图 6-149　下载文件链接



图 6-150　导入文件选择

2）单击“上载”按钮，即可开始上载的动作，如图 6-151 所示。

导入预览

标识	位置
com.myproject.____	pcd:portal_content/com.myproject.proj/com.myproject.IVIEW
com.myproject.Atten	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_attend
com.myproject.AttlnApp	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_attend/com.myproject.f_atteninp
com.myproject.DepotMtApp	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_wsop/com.myproject.f_depotmt
com.myproject.f_attend	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng
com.myproject.f_atteninp	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_attend
com.myproject.f_attenquery	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_attend
com.myproject.f_depotmt	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_wsop
com.myproject.f_fixassdel	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_pm
com.myproject.f_fixassmt	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_pm

Line 1 / 27

图 6-151　导入预览

3）导入完成后，会有导入对象的一个明细清单，导入状态也会显示是否导入成功，如图 6-152 所示。

导入过程已完成。用时：4 秒。错误：0。警告：0。

取消导入　返回到预览面页

导入状态的细节

开始时间	状态	标识	位置
14:55:06	确定	com.myproject.f_wsop	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork
14:55:06	确定	com.myproject.FixAssMtApp	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_pm/com.myproject.f_fixassmt
14:55:06	确定	com.myproject.tspk_0001	pcd:portal_content/com.myproject.New_Folder
14:55:06	确定	com.myproject.f_depotmt	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_wsop
14:55:06	确定	com.myproject.VReqApp	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_reqvac/com.myproject.f_vacreq
14:55:06	确定	com.myproject.f_atteninp	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_attend
14:55:06	确定	com.myproject.f_fixassmt	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_pm
14:55:05	确定	com.myproject.____	pcd:portal_content/com.myproject.proj/com.myproject.IVIEW
14:55:05	确定	com.myproject.f_attend	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng
14:55:05	确定	com.myproject.f_reqvac	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng
14:55:05	确定	com.myproject.FixAssDel	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_pm/com.myproject.f_fixassdel
14:55:05	确定	com.myproject.f_logiwork	pcd:portal_content/com.myproject.proj
14:55:05	确定	com.myproject.AttlnApp	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_attend/com.myproject.f_atteninp
14:55:04	确定	com.myproject.f_fixassdel	pcd:portal_content/com.myproject.proj/com.myproject.f_logiwork/com.myproject.f_pm

图 6-152　导入状态明细

4）接着，在目标系统中切换至内容管理，就能看到所有传输对象已成功创建，传输工作完毕，如图 6-153 所示。

注：如果是 Web Dynpro 页面的传输，除了门户内容的传输之外，Web Dynpro 工程仍然需要通过开发工具或者 NWDI 工作将程序在新服务器中进行部署，这样才能正常运行。



图 6-153　导入后的效果

6.4.3　系统监控

在系统监控中，主要是处理门户内容的锁定。因为门户配置时，系统会对打开或正在编辑的对象自动锁定，通过此处可以查看所有处在锁定模式中的对象，并对处于保护时间内的锁定对象解锁。

进入对象锁定页面，可以看到锁定管理器对应的所有锁定内容，如图 6-154 所示。

可用的锁定管理器：com.sap.portal.pcd

搜索标准：所有锁定的对象

搜索：　搜索

图 6-154　搜索锁定对象

当管理员在编辑门户对象时，清单中就会出现对应的锁定内容。由于用户在操作时不正确地退出或其他原因造成的异常退出，都会使编辑对象始终处于锁定状态。如果需要解锁指定对象，则选中并单击“解锁”按钮即可，如图 6-155 所示。

锁定的对象

解锁　全部刷新

	对象类型	对象标识	位置
☑		com.myproject.AttlnApp	pcd:portal_content/com.myproject.proj/com.myproject.f_hrmng/com.myproject.f_attend/com.myproject.f_atteninp

图 6-155　解锁对象操作界面

第 7 章　SAP 业务流程管理

业务流程管理（Business Process Management，BPM）这一概念的出现是为了让企业能在流程自动化流转的基础上集中化地处理业务，规范员工的业务处理，并提高企业的运转效率。

目前 SAP 的流程管理产品有以下 3 种：一种是运行于 ERP 系统中的工作流（workflow），工作流基于 ABAP 语言进行开发，而且已深度集成在 ERP 系统，因此与 ERP 的逻辑交互具有独到的优势，但是同样受限于 ERP 的 C/S 体系架构，在集成应用与场景运用方面，工作流并不能满足目前日益增多的轻客户端浏览需求，而且对于一些公开技术的支持也有一定缺陷；另外一种是流程整合系统（SAP Process Integration，SAP PI）的流程整合工具，集成在 SAP PI 的开发平台，主要用于多系统间的数据流整合和管理；还有一种就是 NetWeaver 的 BPM 业务流程管理，以用户为核心的流程主要集中在高度交互性的跨组织、跨系统的流程类型，并且将用户的动作与后台服务进行无缝地连接，同时它运行于 Java 架构的 SAP 应用服务器之上。由于 Java 语言以及 J2EE 的优势，SAP BPM 也秉承了跨平台、可移植、分布式等特性。

7.1　BPM 系统架构

要了解 SAP BPM 之前，首先要了解 SAP NetWeaver 版本的变化，在 7.0 版本的企业门户中，如果需要处理用户的业务流程需求，使用的是门户中的向导流程（Guided Procedures，GP）；而到了 7.1 版本，SAP 首先修改了门户的概念，升级成为了 SAP 复合环境（Composite Environment，CE），也就是在使用 SAP BPM 时经常能听到的“CE 系统”；接着从 SAP NetWeaver CE 的 7.1 版本提供的增强包 1（Enhancement Package1，EHP1）开始，这种集成化的复合环境包含了流程的建模、连接、编译、部署和维护复合业务流程所需要的功能，也就是 SAP NetWeaver 业务流程管理（Business Process Management，BPM）。

SAP BPM 作为 SAP NetWeaver 平台的重要组成部分，提供了设计、业务流程建模、实施、运行、流程运行监控的一整套工具，帮助 SAP 的客户机与 SAP 已有的应用模块实现自动化的业务流程流转，跨越不同的系统来控制和提升业务流程的运行效率，能够提高流程的整体透明度，使得企业的复杂的业务流程能够高效率地、更加自动化和更加智能地运行。另外，SAP BPM 还能降低应用软件的开发成本、缩短上市时间、加强法规遵从的贯彻力度、实现业务流程性能的最优化。虽然 BPM 系统的实施并不会大幅地修改原有业务流程，但是与关注业务流程的文档和流程分析结合以后，BPM 就会成为企业改善业务效率的利器。在 BPM 提供的高级开发环境中，它采用流程驱动的模型和性能评估来实现 IT 解决方案的开发。当然，从最终用户的角度来看，BPM 工作流将会把所有的待办事项都主动推送给用户，用户只需要通过浏览器登录并查看门户首页的待办事项，单击打开即可处理，既方便了用户的操作，又提升了工作效率。

SAP BPM 的流程管理主要包含了以下 3 个模块的核心功能：流程设计（Process Compos-

er），集成在 SAP 的开发工作室中，提供了流程建模功能，并且可以通过图形化的设计完成流程的创建和配置；流程桌面（Process Desk），集成在门户中，一方面通过通用工作清单对 BPM 流程待办进行展示，另一方面使用 Web Dynpro 技术开发的页面进行具体的业务操作和系统间的数据传递；流程服务器（Process Server），集成在 NetWeaver 的复合环境服务器中，可以通过 NWA 管理者平台进行管理和监控。流程管理与底层的规则设计、规则管理以及规则引擎搭建出一个完整的 SAP 业务流程管理平台，如图 7-1 所示。

凭借这些功能和架构的优势，BPM 已经逐渐成为 NetWeaver 平台的重要管理工具，并在企业信息化系统的系统集成、流程整合与优化中扮演着至关重要的角色。



图 7-1　业务流程管理架构图

7.2　BPM 流程设计

SAP BPM 的流程设计包含了模型设计、任务配置、流程参数配置以及启动方式配置等几个方面内容。

7.2.1　模型设计

由于 BPM 的模型设计是在基于 Eclipse 的开发工具中直接以图形化的方式来进行的，因此在有条件的项目中可以让业务人员直接参与并完成流程的模型设计。每一个流程中都包含 BPM 流程模型的所有组成要素：事件、活动、网关等多种对象，如图 7-2 所示。当然每一个要素可能包含了各种不同的子要素，这些要素的组合成为复杂流程的原型。

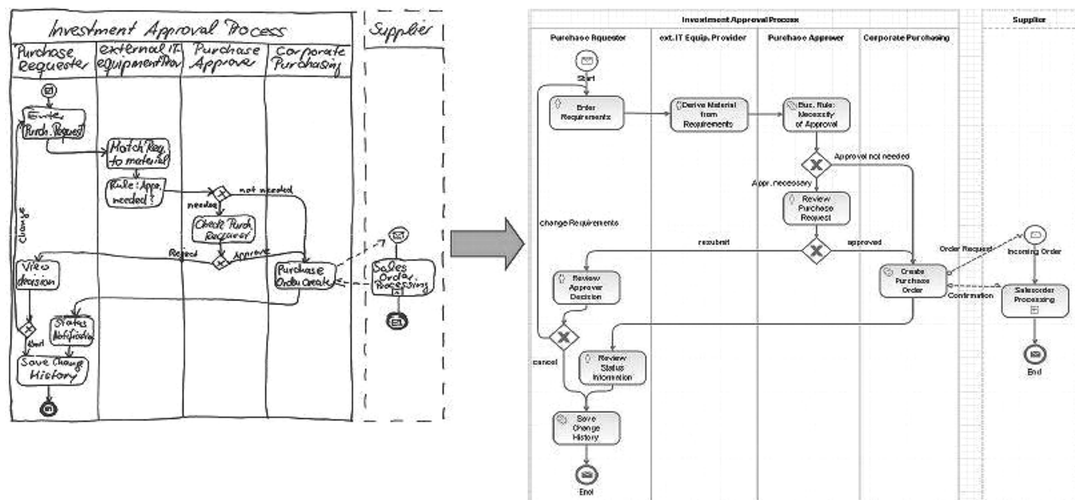


图 7-2　将草图转换为 BPM 流程图

这样的模型设计方式包含了很多的优点：流程的模型设计支持通用的流程对象（事件、任务、角色等），模型的设计可以在 Eclipse 平台并且以图形化方式进行，直观的设计使开发人员或业务人员更方便地上手，流程任务可以方便地集成基于角色的动态视图，也可以实现复杂的审批规则需求（会签、并签等）。

从开发的角度来看，SAP BPM 的流程设计工具已集成在了新版本的开发工具（NWDS）中，可以通过开发工具的切换透视图按钮，方便地切换至流程设计透视图，如图 7–3 所示。

简单地说，整个流程设计的过程包含了以下几个步骤：

1）创建一个流程并进行必要的“泳道”定义。

2）定义流程步骤和对象（任务）。

3）开发并配置流程的启动方式。

4）编译并部署流程至服务器端。

在定义流程对象的过程中，既可以通过模型设计视图中鼠标悬停在流程对象上方时的弹出菜单进行相关对象的定义，也可以通过右侧的流程对象清单进行流程设计，如图 7–4 所示。

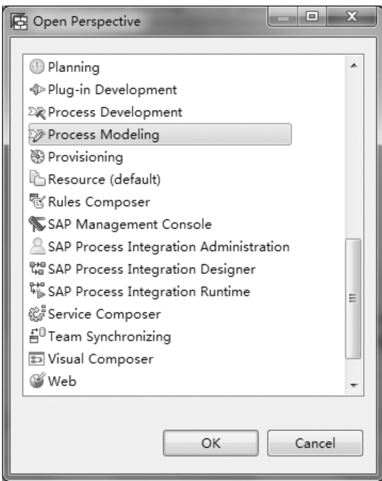


图 7–3　选择透视图

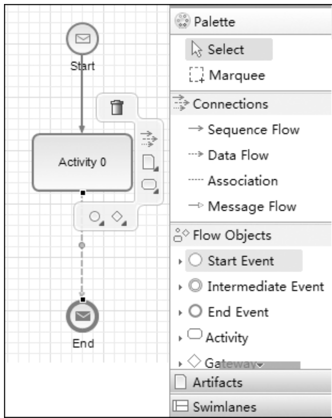


图 7–4　流程建模界面与工具栏

在 BPM 流程设计过程中，主要使用了下列对象。

- 动作（Activity）：即流程中的一个步骤由于表示工作或者动作需要被执行。
- 网关（Gateway）：用于控制流程的走向，包括合并和分散的动作，同时只能通过逻辑控制走向，它本身不能进行走向的判断。
- 事件（Event）：一个标识说明“事件发生”，可以用于启动、暂停、恢复、中断以及重定向一个流程或者动作。

SAP 为了满足不同的业务需求，对于流程设计的这 3 种对象也提供了各种不同的类型，可供开发人员或业务人员进行选择。

动作的类型及其说明见表 7–1。

表 7–1　动作的类型及其说明

动作的类型	图　例	类 型 说 明
Human Activity	Human 0	用户动作，由最终用户进行任务处理

（续）

动作的类型	图　　例	类 型 说 明
Automated Activity	Automated 0	自动动作，由系统自动完成任务处理
Mapping	Mapping 0	一个将流程中上下文进行映射的简单动作，并允许其他接口或规则进行访问
Embedded Sub－Process	Embedded Sub-Process 0 Start Event 0　End Event 0	嵌入的子流程，一个引用并执行另一个流程的动作，触发将进入另一个具体流程中

网关的类型见表 7-2。

表 7-2　网关的类型及其说明

网关的类型	图　　例	类 型 说 明
Exclusive Choice		网关通过一个布尔型的参数表达式进行逻辑判断，仅有一个出口可以被选择
Parallel Split		拆分网关，将 BPM 流程拆分成若干并行的流程继续流转
Parallel Join		合并网关，将若干并行的流程进行合并，与上面拆分的网关相对应
Event based Choice		仅有一个出口可以被选择，但逻辑判断是通过事件来决定，而不是参数表达式
Uncontrolled Merge		将被排除的 BPM 流程进行合并，且选择一个出口继续流转

事件的类型及其说明见表 7-3。

表 7-3　事件的类型及其说明

事 件 类 型	图　　例	类 型 说 明
Start Event		开始事件，作为流程的开始事件（包括一个新的流程或者子流程）
IntermediateTimer Event		时钟事件，暂停一个流程一段时间，或暂停至一个特定时间
Intermediate Message Event		消息事件，需要等待并收到一个消息
Intermediate Error		错误事件，以报错的方式终止一个动作
End Event		结束事件，结束一个流程的事件
Termination		终止事件，立即终止整个流程

除了上述的一些关键对象之外，为了流程能够让人一目了然，模型设计中还提供了“泳道”的创建，通过“泳道”可以直观地区分出流程中不同任务的执行主体，即对应的任务处理部门或岗位。

7.2.2　任务定义

从某种意义来说，模型的设计只是为流程定义了一个流转的框架，经过对实际业务流程的分析，大部分的任务步骤都是通过“用户动作”（Human Activity）操作完成的，如审核和确认等动作。从逻辑角度来看，用户动作的内容是通过任务的定义和配置来实现的；而从用户的使用角度来看，每一个任务对应的就是每一个用户在登录客户端系统后看到的“待办任务”。

使用开发工具可以很方便地在一个 BPM 工程中，通过右击的新建菜单创建一个流程以及对应结点的任务，如图 7-5 所示。

1. 属性配置

当打开一个任务时，会看到任务包含了概览、特性和动作、角色、用户文本以及时间常量等几个选项卡，如图 7-6 所示。

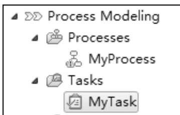


图 7-5　流程模型目录结构

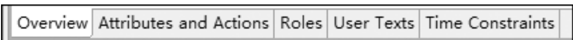


图 7-6　不同的选项卡

- 在概览中，可以定义任务的基本信息，可以选择是否需要邮件通知对应任务的待办接收人，可以选择当前任务的优先级（将显示在用户的待办清单中），可以绑定任务对应 Web Dynpro 程序的接口视图，但视图中必须有事件触发动作，用于触发工作流继续流转。由于 Web Dynpro 程序决定了用户在打开待办时可以看到以及处理业务时的页面，因此必须绑定，如图 7-7 所示。
- 在特性和动作中，可以配置自定义的特性，并且可以使用一些模板提供的逻辑表达式，还可以配置自定义的动作，如图 7-8 所示。

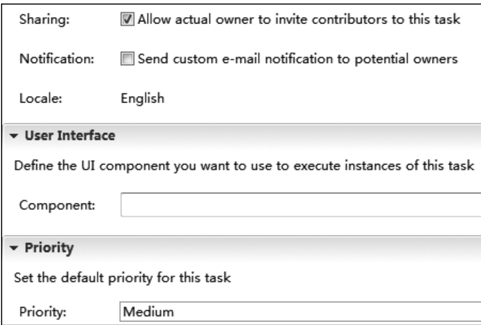


图 7-7　概览选项卡

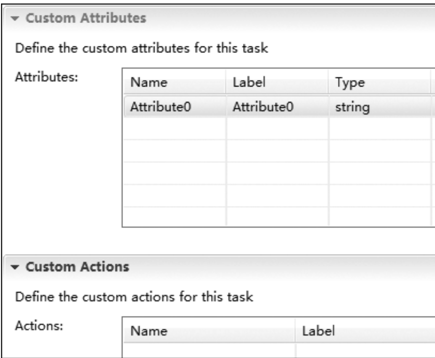


图 7-8　特性和动作选项卡

- 在角色中，可以定义当前任务的处理人、定义例外用户以及定义任务的管理员，如图 7-9 所示。

- 在用户文本中，可以定义一些特定文本的变量，用于任务的名称或者说明信息中，当然这些文本也可以使用流程中传递的上下文特性作为参数，动态地生成特殊文本，如图 7–10 所示。

图 7–9　拥有者选项卡

图 7–10　用户文本选项卡

- 在时间常量中，可以指定任务的延迟触发时间以及任务的过期时间，避免用户收件箱中出现待办任务过量堆积，如图 7–11 所示。

2. 所有者配置

在任务的属性配置中，Web Dynpro 接口视图的定义非常重要，因为视图的绑定决定了用户处理业务的页面选择以及实际体验；而在确定接口视图之后，最为重要的则是任务的所有者配置，因为所有者的配置决定了哪些人有可能会收到任务、哪些人不会收到任务以及哪些人有权限可以对任务进行终止或转发，即分为了“潜在所有者”“排除所有者”以及“任务管理者”3 种类型，如图 7–12 所示。

图 7–11　时间常量选项卡

图 7–12　所有者配置

在 3 种类型的所有者配置选项卡，都包含了以下两种方式进行所有者的指派：“Choose one or more UME principles”，可以理解为 UME 实体；“Use an expression”使用表达式。在选中“Choose one or more UME principals”单选按钮后，发现其实不仅可以绑定用户，还可以绑定指定的组、角色或用户，如图 7–13 所示。

除了绑定静态的 UME 实体以外，SAP 也允许通过表达式动态获取审批人。而在各类的实施项目中，发现绑定静态对象的情况很少存在，这可能是因为配置的不灵活而带来较大的工作量，容错率非常低。相对而言，表达式的情况有多种，一种是将整个审批规则封装成一个过程，在表达式中直接调用这个过程，常见的如使用 EJB 实体；另外一种是表达式直接读取流程上下文中的某一个变量，这个变量已经存在了审批规则过程调用后的结果。

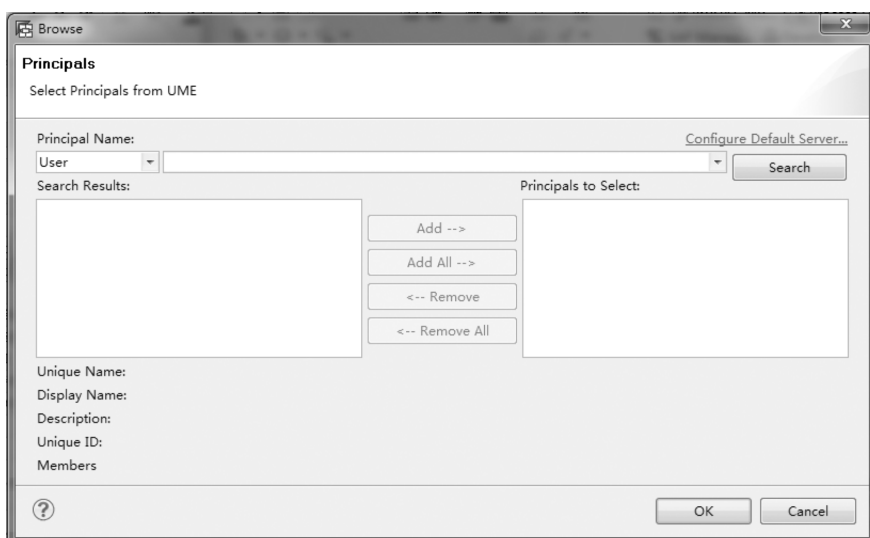


图 7-13　配置 UME 实体界面

直接调用过程的形式将整个审批规则工作配置化，实施者只需要配置任务属性而省去了代码级别的开发。绑定变量的方法简单、直接，但调用规则的过程却需要 Web Dynpro 程序来实现，从一定程度上增加了项目开发的工作量。由于通过 Task 直接调用过程，往往会遇到不能回滚的尴尬，当新的任务生成时会执行通过规则获取审批人的逻辑，一旦执行当中出现异常，新的任务就可能会生成失败。最终的结果是即使能够通过后台监控错误原因，但流程实例本身也会因为异常不能回滚，这种异常情况往往对流程任务处理者来讲是致命的，他们通过表象不能判断流程的实际走向，最终的处理方式也只能是重新提报等。相反，直接绑定变量的方式却因为代码级别的原因使开发人员能够合理地处理异常，回滚已经执行的操作。

当然不管是通过表达式调用服务，还是绑定某一个动态的执行者变量，区别仅在于执行审批规则逻辑的时间顺序，执行审批规则的过程是必须要有的。审批规则对应于流程任务处理者的取人逻辑，每一个任务生成之前都要通过适应于客户需求本身的审批规则，而现在人们已经把这个过程模块化，模块置于任务生成之前，模块的输出是该任务精确的执行者。值得一提的是，每一个规则背后都包含了对该客户或公司组织架构的深层次理解与分析，对于不同的公司、不同的业务模块来讲，它们的审批规则都是不同的，或者说人们还无法抽象出一种规则，使它应用于所有的业务场景，但规则的设计却有一个最终的目的——配置化的灵活、简单与扩展。当企业的组织架构出现调整，当员工的岗位级别发生变动时，规则能够通过简单的配置来灵活地应对这种变化，使 BPM 流程本身也具备更强的业务兼容性。

下面归纳了几种常用的设计原型：

1）自定义配置审批人：通过自定义配置表来配置每一个流程结点的审批人，配置表直接指定具体岗位或人员。配置表的属性一般包含流程标识（ID），任务标识，所有与审批相关的业务类型、业务数据字段，员工标识。对于一个具体的流程实例，通过已知的流程标识、任务标识、业务类型涉及的业务数据来获取相关的审批人。这种方法适用于大部分通过业务类型数据不同指定不同审批人的情况，并且配置表已经具体到员工标识，通过配置表可以实现较多特殊情况的审批规则，配置较为灵活，但由于配置的过程对所有细分的情况都有涵盖，初始化的配置会带来很大的工作量，而且对于审批规则调整的适应性不高，企业人员

体系的变化或者岗位调动都会给维护带来较大的工作量。

2）通过角色获取审批人： 通过角色的方式配置审批规则，相当于对审批级别做了分类，在整体审批级别层次较为清晰、审批层次对应的人员集合较为固定的情况下比较适用。例如，一个集团所有流程的审批最多就是三级，三个级别对应了固定的专岗人员，那就可以建立三个角色，每一个角色导入这些专岗人员。实际的审批过程是从角色当中取人，然后针对这些人员组合做进一步过滤。这种方法的优点是整个配置过程简单，初始化与维护都不会有很大工作量，但这种方式只是针对较为简单的审批场景，对于复杂的业务类型审批往往不能够满足要求。

3）与 HR 系统（模块）的集成： 审批规则的设计与 ERP 中的 HR 系统集成是比较好的一种方式，一方面，由于审批人的数据源就是来源于 HR 本身，因此不用关心因为人员级别的变化而带来的影响；另一方面，HR 已有的数据配置也大大减少了初始化的配置工作量。例如，企业本身实施的 SAP 人力资源模块（Human Resource，HR），针对流程标识，任务标识的配置可能就是“组织－职位－员工”（Organisations－Positions－Employees，O－S－P）的一种，在不指定具体员工的情况下，组织和职位下的人员都来自 SAP HR 数据，组织或岗位下的人员调整都不会对流程的实时审批产生影响。当然采用这种设计的前提是要对企业的 HR 系统有很好的调研与理解，而且审批规则确实与 HR 的体系关联很大，同时 HR 提供系统接口也是必然，这样会带来一定的接口集成的工作。

7.2.3 启动方式选择

不同的 BPM 流程可以有多种启动方式，如管理者平台的手动触发、Web 服务启动，API 调用的方式启动等方式。通过流程控制台手工启动流程适用于对基础流程的场景测试，而后两种则是从开发的角度发起流程实例，如图 7-14 所示。

图 7-14　BPM 启动事件触发器的配置

比较开发端两种启动流程的方式，Web 服务的方式从接口的角度来讲更通用也更规范。对于其他系统来讲，通过调用流程的 Web 服务就可以发起 BPM 流程，但客户端调用的过程却较为烦琐。对于每一个 Web 服务都要有一个生成客户端代理并调用服务的过程，而且如果是 Web Dynpro 调用 Web 服务，当流程的启动接口结构发变化时，往往要重新导入模型，使因为接口结构的变化带来了客户端程序的调整，同时由于 BPM 发布后生成的 Web 服务本身是异步的机制，不能通过服务返回的参数来判断或获取流程相关的信息（如标准流程 ID），这些数据对于标准的流程处理（如取消流程等操作）都是必须的，因此通过 WS 启动流程的种种弊端让大部分实施者在后期实施中采用 API 的方式来启动流程。比较前者相对于规范化的启动接口来讲，API 的启动方法可以适用于所有流程，接口结构的变动不会对方法带来太大的影响。另外，API 具有返回参数，参数中包含了 URI 形式的标准流程 ID，这些都会让最终的调用过程变得高效，而且功能更加全面。总的来讲，对于系统间的服务调用来发起流程，采用 WS 是比较好的一种方式，对于系统本身的流程启动更多的是倾向于标准 API。

1. 手动触发

使用管理员账号登录 NWA 平台，在菜单中选择“Configuration”，随后单击“Processes and Tasks”就能打开流程和任务的管理界面，如图 7-15 所示。

在清单中列出了所有成功部署至服务器的流程，单击选中任一流程，在下方任务清单中选中开始事件，随后单击“Start Process”按钮，就能从服务器端手动发起指定流程。但由于这种方式不够灵活，且必须由管理员手动进行操作，因此在实际项目应用中较少使用。

2. Web 服务启动

一般在定制流程模型、开始事件时，都会关联对应的服务接口（Service Interface），即流程的启动服务接口（也是流程发布服务器后 Web 服务的访问接口），对应于 Web 服务的 WSDL 地址，如图 7-16 所示。

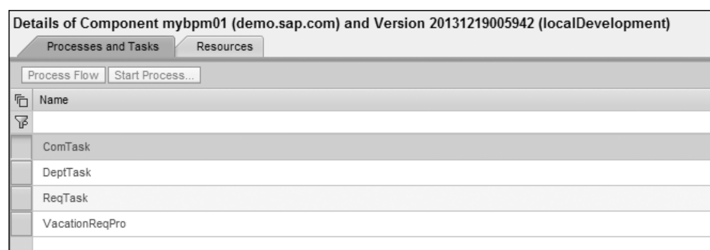


图 7-15　流程中的任务清单

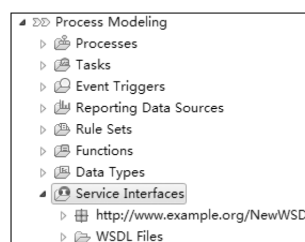


图 7-16　项目目录结构

从服务定义的角度来说，BPM 在部署到服务器后都会产生一个对应的 Web 服务，其中包含发起流程的方法以及必要参数，通过调用该 Web 服务触发相应的事件就能启动流程，并传入必要参数的值。

开发实例：

1）通过服务接口的关键参数在管理员平台的“WS Navigator”进行测试，以保证在使用时可以正常运行。进入 NWA 页面的“SingleService”服务，并输入关键字进行搜索，如图 7-17 所示。

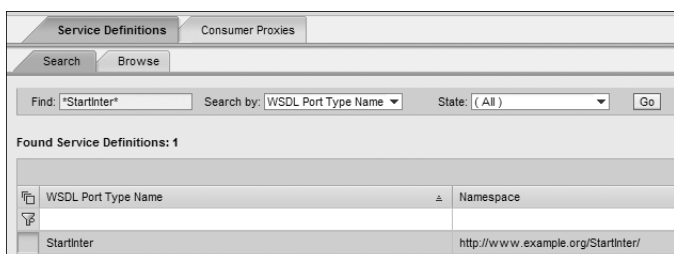


图 7-17　搜索服务界面

2）查找到对应服务后，单击下方的“WSDLs”页签，并单击“Test”按钮，如图 7-18 所示。

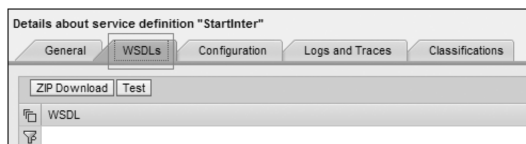


图 7-18　WSDL 界面

3）随后跳转至 Web 服务测试页面“WS Navigator”，单击“下一步”按钮，如图 7-19 所示。

4）输入必要的执行方法参数，单击“下一步”按钮，如图 7-20 所示。



图 7-19　选择对应操作



图 7-20　输入参数值

5）执行完成后，通过执行返回消息确认服务执行成功，如图 7-21 所示。

6）接下来通过在 Web Dynpro 工程中模型导入 Web 服务，并创建对应的执行方法或生成自定义控制器进行执行，便能达到代码端启动流程的目的，如图 7-22 所示。

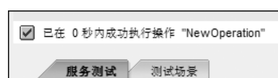


图 7-21　测试结果

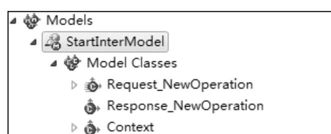


图 7-22　导入模型

3. 流程 API 启动

通过 API 启动流程是在 NetWeaver 7.2 及更新版本中提供的更受开发人员欢迎的一种方式，因为它相对于 Web 服务的定义和使用来说，更加地简洁和直接。值得一提的是，在新版本中 SAP 不仅公开了 BPM 启动流程的 API，还提供了对操作任务的 API 以及操作流程的 API 等内容的支持，这些 API 使开发者实现各种各样的增强（包括对流程的动态控制）成为了可能，有时通过 API 的使用也能达到一些意想不到的效果。

使用 API 启动流程的代码示例：

```
//@ @ begin startProcess( )
    try{
        logger. infoT( "标准 API 启动流程:" ) ;
        // 获取流程定义管理器
        ProcessDefinitionManager processDefinitionManager = BPMFactory
                . getProcessDefinitionManager( ) ;
        // 获取流程的定义,赋值流程技术名称 vendor dcname processname
        ProcessDefinition activeProcessDefinition =
          processDefinitionManager
        . getActiveProcessDefinition( "demo. sap. com" , "mybpm01" ,
          "VacationReqPro" ) ;
        if( activeProcessDefinition == null) {
        logger. errorT( "标准 API 启动流程,启动失败" ) ;
        return;
        }
        // 获取流程启动管理器
        ProcessStartManager processStartManager = BPMFactory
        . getProcessStartManager( ) ;
```

```
// 获取流程启动事件
Set < ProcessStartEvent > processStartEvents = processStartManager
. getProcessStartEvents( activeProcessDefinition. getId( ) ) ;
ProcessStartEvent processStartEvent =
   processStartEvents. iterator( ). next( ) ;
// 创建流程启动入口参数
DataObject doStart = processStartManager
. createDataObjectForStartEvent( processStartEvent) ;
// 根据 StartInterface 入口参数逐级创建 DataObject
DataObject baseInputDO = doStart. createDataObject( "BusiData" ) ;
// 发起人
baseInputDO. setString( "startUser" , "test00" ) ;
// 发起人部门
baseInputDO. setString( "startUserDept" , "1001" ) ;
// 下级审批人 unique id
baseInputDO. setString( "nextApprPers" ,
   "USER. PRIVATE_DATASOURCE. un:test01" ) ;
// process pool name
baseInputDO. setString( "poolName" , "请假申请" ) ;
// 下一级结点,结点号根据流程配置动态获取
baseInputDO. setString( "taskNode" , "20" ) ;
// 公司代码(发起人)
baseInputDO. setString( "bukrs" , "0001" ) ;
// 网关
baseInputDO. setString( "gateWay" , "" ) ;
// 产生的自定义表流程 ID,可以定义各种生成规则,保证并发唯一,此处为模拟生成
String processId = "P20130000000000001" ;
// 任务名称,标题根据流程配置动态获取
baseInputDO. setString( "taskName" , processId + "请假申请一级审核" ) ;
// roadMap 的字符串存储
baseInputDO. setString( "roadMapStr" , "" ) ;
// 操作事务码
baseInputDO. setString( "transacCode" , "" ) ;
// autoID(存表时自动生成主键)
baseInputDO. setString( "metAutoId" , autoId) ;
// 流程流水号
baseInputDO. setString( "procNumber" , proNumber) ;
// 流程所属流程定义码
baseInputDO. setString( "processCode" , "P0001" ) ;
baseInputDO. setString( "taskNumber" , "" ) ;
doStart. set( "BusiData" , baseInputDO) ;
// 启动流程
URI startProcess = processStartManager. startProcess(
         processStartEvent, doStart) ;
logger. infoT( "启动流程成功!" ) ;
// 流程启动成功返回 URI 和流程的 ProcessInstanceId
String uri = startProcess. toString( ) ;
// uri 转化为字符串格式
   // bpm://bpm. sap. com/process - instance/ + saptaskid
String[ ] split = uri. split( "process - instance/" ) ;
// 标准流程 ID 取分隔符最后部分
wdComponentAPI. getMessageManager( ). reportSuccess( "Process Start
   Success!" ) ;
return;
} catch( Exception e) {
```

```
            logger. errorT("流程启动失败" + e. getMessage());
            return;
        }
        //@@ end
```

注：这段 Web Dynpro 中的代码只有添加对“tc/bpem/facade/ear”与“tc/je/sdo21/api”两个标准软件组件的依赖才能编译通过。

7.3 BPM 流程桌面

SAP 的 BPM 流程桌面主要包含以下几个功能：

- 通用工作清单，作为所有待办任务、提醒事件以及通知消息的统一收件箱。
- 待办处理框架，在 Web Dynpro 工程页面之外提供了一些流程处理的标准功能（不需要进行 Web Dynpro 的额外开发）。
- 门户中的流程清单视图，用于显示具体流程以及结点的相关信息，方便用户对流程进行一定程度的监控。

7.3.1 通用工作清单

1. 通用工作清单配置

通用工作清单（Universal Worklist，UWL）是 SAP NetWeaver 架构之上系统的待办工作平台。通过配置不同系统的连接器，UWL 可以集中地显示多个平台不同的工作流待办，包括流程任务、警告、提醒等内容。通常情况下，通用工作清单更多地应用于 BPM 的待办清单，用户登录 SAP 企业门户后就可以查看属于自己的待办事项，并通过 UWL 提供的功能查看待办明细。单击待办标题，也可以打开对应的业务处理页面（包括 Web Dynpro 开发页面、ERP 系统的 GUI 页面或是其他第三方系统的页面）处理相关业务，如图 7-23 所示。

图 7-23 通用工作清单

从配置角度来看，通用工作清单是企业门户中一个默认提供的门户视图，参考中文目录在门户内容中的“门户内容/由 SAP 提供的内容/最终用户目录/标准门户用户/iView/com. sap. netweaver. bc. uwl. iviews/通用工作清单”。门户配置人员可以通过查看或修改该视图的属性对通用工作清单进行一些简单个性化设置，如图 7-24 所示。

图 7-24 通用工作清单的属性

在系统的标准角色中，用户核心角色“pcd:portal_content/every_user/general/eu_core_role”已经添加了通用工作清单视图的增量链接，因此普通用户添加了该角色，就可以在门户页面中

看到通用工作清单。对于一些个性化的用户需求，也可以新建自定义的门户角色，并添加UWL视图形成自己的主页目录结构，如图7-25所示。



图7-25　标准角色配置

通用工作清单作为工作项目收件箱或任务列表，是用户访问工作流“TODO”列表的渠道，包含了下列优点：

1）“统一”的收件箱，接收所有的任务。

2）使用单一固定的处理页面。

3）方便使用，同时也支持客户化的定制。

4）支持多渠道访问。

5）与SAP NetWeaver平台集成，通过门户集中访问。

2. 通用工作清单API实例

SAP在新版本中已经公开了部分UWL的API，以供开发者通过代码的方式读取待办数据，同时通过BPEM开发包中的“TaskInstanceManager”类，也可以读取待办。

从实际调用的情况来看，UWL的API构造的各种参数较为复杂，需要对方法的输入/输出结构有很好的掌握。相对而言，使用BPEM的API调用的过程比较简单，方法默认做了权限处理，只能查询登录用户的待办，对于大数据量的待办清单查询，BPEM API的效率要优于前者。下面分别通过两个实例具体介绍两种方法的使用和相关代码。

开发实例：

1）编辑任意一个Web Dynpro工程的视图，创建两个按钮并创建按钮的默认动作，如图7-26所示。



图7-26　运行效果

2）使用UWLAPI的参考代码：

```
try{
Properties env = new Properties();
env.put(InitialContext.INITIAL_CONTEXT_FACTORY,
"com.sapportals.portal.prt.registry.PortalRegistryFactory");
//获取uwlservice
InitialContext ctx = new InitialContext(env);
IUWLService uwlService = (IUWLService) ctx
.lookup("/broker/services/" + IUWLService.ALIAS_KEY);
//设置访问超时时间
final int sessionIdleTimeout = 600;
UWLContext uwlContext = new UWLContext();
//赋值当前登录账户
IWDClientUser clientUser = WDClientUser.getCurrentUser();
IUser user = clientUser.getSAPUser();
uwlContext.setUser(user);
uwlContext.setLocale(Locale.getDefault());
```

```
//启动访问会话
IUWLSession uwlSession;
uwlSession = uwlService
. beginSession( uwlContext, sessionIdleTimeout) ;
uwlContext. setSession( uwlSession) ;
//获取 itemManager
IUWLItemManager itemManager = uwlService. getItemManager( uwlContext) ;
//获取 uwlView
UWLView uwlView = uwlService. getViewManager( uwlContext)
. getViewForItemType( ItemType. UWL_ITEM, uwlContext) ;
QueryProperties queryProperties = new QueryProperties( ) ;
//设置返回最大条目数
queryProperties. setMaxNumberOfItemsToFetch(300) ;
//定义排序字段
String[ ] sortBy = {Item. CREATED_DATE } ;
queryProperties. setSortBy( sortBy) ;
boolean[ ] sortAscending = {true } ;
queryProperties. setSortAscending( sortAscending) ;
//调用 API 获取结果集
QueryResult qrs = itemManager. refreshCacheAndGetItems( uwlContext,
uwlView, queryProperties, null) ;
ItemCollection items = qrs. getItems( ) ;
Item item = null;
//遍历返回
for( int i = items. size( ) - 1;i  >= 0;i -- ) {
   item = items. get( i) ;
   String subject = item. getSubject( ) ;//任务标题
   this. wdComponentAPI. getMessageManager( ). reportSuccess( subject) ;//打印标题
   }
} catch( NamingException e) {
   this. wdComponentAPI. getMessageManager( ). reportException( e) ;
} catch( Exception e) {
   this. wdComponentAPI. getMessageManager( ). reportException( e) ;
}
```

3）部署并运行应用后测试按钮，打印的内容与待办清单的显示条目一致，如图 7-27 所示。

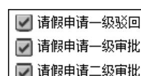


图 7-27　提示消息

4）第二个按钮使用 BPEM API，参考代码如下：

```
TaskInstanceManager taskInstanceManager;
try{
   taskInstanceManager = BPMFactory. getTaskInstanceManager( ) ;
   HashSet < Status > statuses = new HashSet < Status > ( ) ;
   statuses. add( Status. READY) ;
   statuses. add( Status. RESERVED) ;
   statuses. add( Status. IN_PROGRESS) ;
   Set < TaskAbstract > myTasks = taskInstanceManager
   . getMyTaskAbstracts( statuses) ;
   for( TaskAbstract task ： myTasks) {
      wdComponentAPI. getMessageManager( ). reportSuccess( task. getPresentationName( ) ;
   }
} catch( Exception e) {
   wdComponentAPI. getMessageManager( ). reportException( e. getMessage( ) ) ;
```

```
        logger. errorT( e. getMessage( ) );
    }
```

5）部署并运行应用后测试按钮，打印的内容与“UWL API”按钮一致，条目没有排序，如图 7-28 所示。

注： 使用 API 的 Web Dynpro 工程，只有依赖 UWL 与 BPEM 的标准组件包，才能引用相关的 API 并调用，包括“tc/kmc/bc. uwl/api”“tc/bpem/facade/ear”以及“tc/je/usermanagement/api”，如图 7-29 所示。

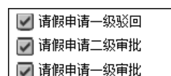


图 7-28　提示消息

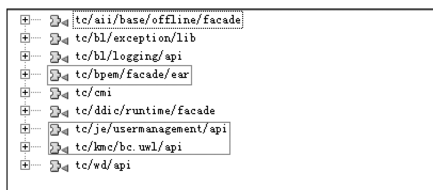


图 7-29　需要依赖的开发组件

3. 客户化工作清单

由于实际的项目中客户往往提出大量的个性化、定制化的需求，如待办清单中需要显示一些业务数据等，这样 SAP 标准的通用待办清单的功能就大大受限，需要自定义的待办清单来适应这种增强。通过平台提供的 API，可以从代码端获取后台的待办数据，并自定义待办清单来满足特殊的业务场景。

开发实例：

1）自定义开发的工作清单功能区。图 7-30 中的工作清单通过 API 读取并打印了待办标题。BPM 中的待办标题是可以通过变量的形式动态设置的，动态设置的标题存储于标准表中，通过标题中的关键字可以实现标准表与自定义表的关联，从而显示与待办相关的业务数据，如图 7-30 所示。

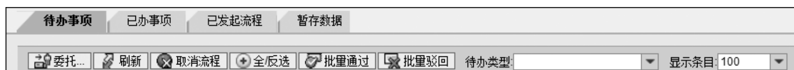


图 7-30　自定义工作清单示例

2）自定义开发的工作清单内容区。数据库设计中会定义一张与标准表实例一一对应的流程表。自定义的流程表中可以包括除流程属性外的所有业务关键数据，当流程启动时，会在自定义表中插入一条流程数据，同时流程主键会随着流程上下文在流程中传递，每一个待办任务结束并产生下一个待办时，流程主键都会包含在新的待办标题中，最终通过解析标题数据得到流程关键值，并关联自定义表将流程属性与业务数据显示在通过 Web Dynpro 开发的自定义工作清单中。页面元素通过选项卡集以及表格的形式对待办清单以及待办任务重点信息进行展示，如图 7-31 所示。

待办事项　已办事项

标题	公司代码	部门	流程ID	标准任务ID
@subject	@bukrs	@dept	@processid	@saptaskid
@subject	@bukrs	@dept	@processid	@saptaskid
@subject	@bukrs	@dept	@processid	@saptaskid

图 7-31　自定义流程表示例

3）其中标题使用的是流程流转中的动态变量，即在任务的属性定义中将任务名称修改为变量“｛taskName｝”，而在上方变量定义处，将该变量与流程“UIRequest”中上下文的特性进行映射，任务的名称（待办的标题）就可以通过程序的上下文进行自定义设置，如图 7-32 所示。

AppTask
User Texts
Variables
Define the variables you want to use to parameterize the user texts below

Name	Type	Expression
taskName	string	UIRequest/Context/ProcessNode/BusiData/taskName

Parameterized Texts
Define the text that will be sent as an e-mail notification and displayed in the user's inbox. Use braces ({}) to reference t
Subject: {taskName}
Description:

图 7-32　自定义任务配置

4）另外，流程清单中的公司代码、部门代码通过与自定义表的数据关联获取，流程标识是自定义流程实例表的主键值，而标准任务标识是 BPM 标准表中生成的任务主键值，通过 BPEM 的 API 在初始化时编写待办获取逻辑，参考代码如下：

```
IUwldataNode nodeUwldata = wdContext. nodeUwldata( );
IUwldataElement createAndAddUwldataElement = null;
TaskInstanceManager taskInstanceManager;
try{
    taskInstanceManager = BPMFactory. getTaskInstanceManager( );
    HashSet<Status> statuses = new HashSet<Status>( );
    statuses. add( Status. READY);
    statuses. add( Status. RESERVED);
    statuses. add( Status. IN_PROGRESS);
    Set<TaskAbstract> myTasks = taskInstanceManager
    . getMyTaskAbstracts( statuses);
    for( TaskAbstract task : myTasks){
    //通过 URI 获取标准任务 ID
    URI uri = task. getId( );
    String sapTaskId = "";
    if( uri != null){
    String strUri = uri. toString( );
    sapTaskId = strUri. substring( strUri. lastIndexOf( "/") +1,
    strUri. length( ));
    }
    //获取 Task Subject
    String subject = task. getPresentationName( );
    if( subject == null)
    subject = "(无主题)";
    //解析标题获取流程中传递的 PROCESSID
    String processId = "";
    if( subject!= null&&subject. length( ) >18){
```

```
            //自定义标题字符串拼接规范[PROCESSID(18) + 标题内容]
            processId = subject.substring(0, 18);
            }
              //通过 PROCESSID 关联自定义流程表,获取公司、部门
            String bukrs = "";
            String dept = "";
            /**
             * String sql = "";
             * sql = "select bukrs,dept from t_process where processid =
                 ' " + processId + " ";
             *通过 Connection 执行的 sql resultset
             *注:严格意义讲,此处是调用数据库的操作,即通过封装的 Dao 逻辑查询流程的公司、
        部门。
             *此处直接模拟了这一过程
             */
            bukrs = "0001";
            dept   = "1001";
            //通过标准 API 获取的数据构造自定义清单数据
            createAndAddUwldataElement = nodeUwldata.createAndAddUwldataElement();
            //赋值 PROCESSID
            createAndAddUwldataElement.setProcessid(processId);
            //赋值 SAPTASKID
            createAndAddUwldataElement.setSaptaskid(sapTaskId);
            //赋值标题
            createAndAddUwldataElement.setSubject(subject);
            //赋值公司
            createAndAddUwldataElement.setBukrs(bukrs);
            //赋值部门
            createAndAddUwldataElement.setDept(dept);
            }
        }catch(Exception e){
            wdComponentAPI.getMessageManager().reportException(e.getMessage());
            logger.errorT(e.getMessage());
        }
```

代码说明：从编写的逻辑上可以看出，每遍历一次通过 API 读取的结果集都要通过建立数据库连接，访问与数据库获取流程相关的业务数据，当数据量较大时，程序运行效率会降低。由于数据源的特殊性使得开发人员无法保证能够直接通过数据表关联的形式读取数据，因此考虑建立临时表或主键值集合的形式去优化整个查询过程是一个合理的方案。若采用临时表的方案，逻辑处理更为简洁，效率更高，但需要额外的表空间，是典型的通过空间换取时间的做法。相关开发步骤如下：

1）调用 API 并循环结果集时，只是往临时表里批量插入数据，数据包括标准表信息。

2）通过临时表直接关联查询自定义流程表并填充自定义待办清单对应的结点数据，采用集合的形式在第一步通过字符串拼接流程 ID，同时填充已有的数据至结点，随后通过 SQL IN 的集合查询方法一次性查询所有业务数据。

3）再次遍历 Node，并赋值通过第二步批量查询的流程数据。

通过上面的步骤可以看到，建立临时表的方式在步骤上比之前分析的两种方式都简单，但效率却是最低，最后一种集合查询的方法对于数据量不是很大的情况下可以使用。另外，从上面的例子中可以看出，通过 BPM 的 API 读取的待办数据并没有按照某一条件进行排序，

对于排序的需求可以根据不同的查询方式，通过 SQL 语句中的排序逻辑来实现。

4）由于流程的标题通过程序的上下文进行定义，因此在流程启动时千万不要忘记初始化流程标题，并且每一级结束的按钮方法代码中根据流程启动配置更新任务标题，如图 7-33 所示。

```
// 产生的自定义表流程ID，可以定义各种生成规则，保证并发唯一，此处为模拟生成
String processId = "P20130000000000001";
// 任务名称,标题根据流程配置动态获取
baseInputDO.setString("taskName", processId+"请假申请一级审核");
// roadMap 的字符串存储
baseInputDO.setString("roadMapStr", "");
```

图 7-33　自定义代码

5）实例运行后的待办清单，如图 7-34 所示。

待办事项　已办事项

标题	公司代码	部门	流程ID
请假申请一级审核	0001	1001	P20130000000000007

图 7-34　运行效果

7.3.2　任务执行

当用户单击任意一个待办任务时，将会在浏览器中打开一个新窗口并进入对应的任务处理页面。任务处理页面除了在任务中绑定的 Web Dynpro 接口视图或者其他系统的集成页面外，还有一个 SAP BPM 提供的标准 BPM 任务处理框架，如图 7-35 所示。



图 7-35　标准任务处理框架

可以看到，整个页面的最上方是任务标题栏，用于显示任务的标题、开始时间、状态、优先级等信息；在任务栏下方是工具栏，包含了按钮以及标准功能链接，链接分别用于打开附件区以及注释区（默认是关闭状态，在单击链接后才会显示）；整个截图只有左下方才是实际的应用区（图中是 Web Dynpro 的接口视图对应内容的显示）。

其中工具栏包含了 3 个标准功能按钮。

1）“关闭”按钮：仅关闭当前处理页面，不是关闭流程，待办仍然留在待办清单中；但是在关闭后任务的状态会发生变化，从“新（New）”变成了“进行中（In Progress）”。

2）“委派”按钮：将任务委派给一个指定用户，但是指定的对象必须是任务的执行人；如果想要长期委派，如审批人需要出差一段时间无法登录系统，则需要在通用工作清单中进行长期自动委派的设置。

3）“撤销”按钮：需要注意的是，同一个任务仅能由一个用户进行处理（并行分支中的不同任务可以由多用户处理），一旦一个用户打开，其他可处理用户中的待办也会消失，因为系统认为任务已经被“认领”了。所以如果用户在打开任务后发现自己无法处理，则需要单击“撤销”按钮，这样其他处理用户才能在工作清单中重新看到该任务。

7.3.3 流程监控

使用管理员账号登录 NWA 平台，在菜单中选择“Configuration”，随后单击“Processes and Tasks”就能打开流程和任务的管理界面，如图 7-36 所示。

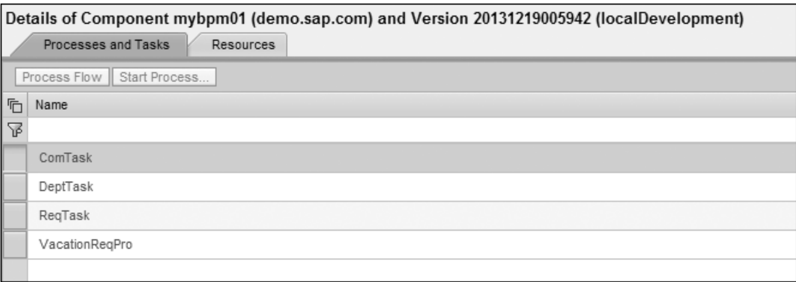


图 7-36　流程管理中的任务清单

选中需要查看的流程，单击“Show Process Flow”按钮，如图 7-37 所示。

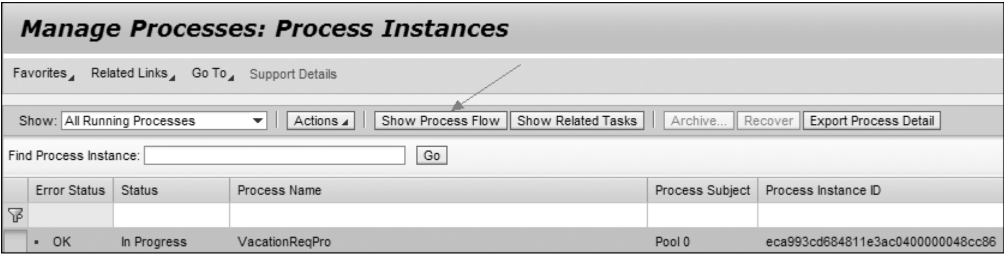


图 7-37　流程实例操作页面

在弹出的页面中就可以查看运行时的流程流转图，根据上面流程的运转情况可以知道该流程处于二级审批，如图 7-38 所示。

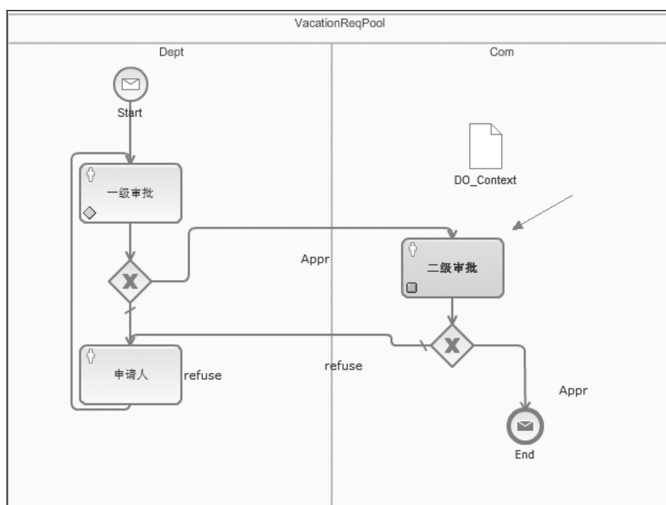


图 7-38　标准的流程流转图

7.4　BPM 开发实例

在简单地了解了 BPM 的组成部分后，下面通过一个实际的例子介绍 WD4J 与 BPM 集成的开发配置过程。以请假申请为例，该流程为二级审批，其中一级为部门领导，二级为公司领导，根据流程审批级次建立流程模型以及必要的 Web Dynpro 开发。

简要的开发流程如下：

1）创建流程开发组件，设计流程模型并定义流程任务。

2）创建 WD4J 项目，定义上下文及流程流转触发事件，并与 BPM 流程进行关联。

3）在 BPM 流程中对流程内部的上下文映射进行定义和配置。

4）对 BPM 流程的启动方式进行具体配置和调用开发。

5）部署 BPM 和 Web Dynpro 项目，运行并测试流程，确认流转和处理是否正确。

7.4.1　创建 BPM 流程

1）打开开发工具 NWDS，选择“File”→“New”→“Project”（见图 7-39），创建一个工程，或单击“新建项目”按钮。

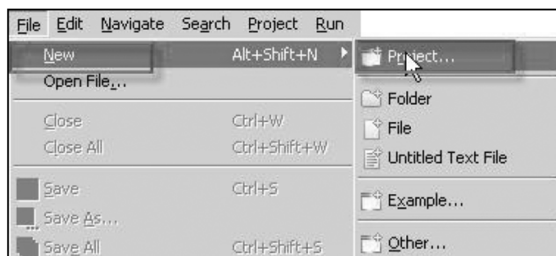


图 7-39　创建项目的菜单选择

2）在弹出的对话框中，选择流程模型类型的开发组件，并单击“Next”按钮，如图 7-40 所示。

3）随后选择本地的软件组件目录，选中“MyComponents”，并单击“Next”按钮，如图 7-41 所示。

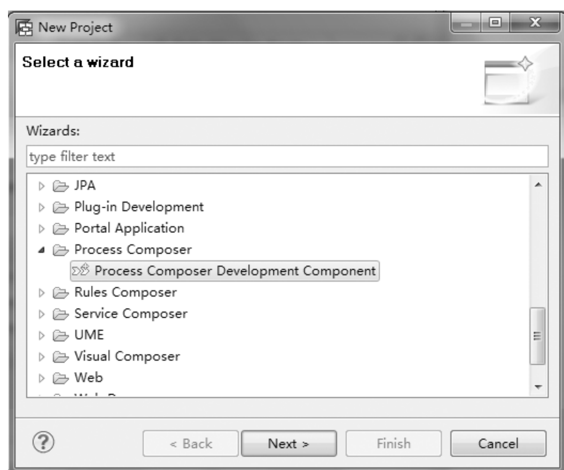


图 7-40　选择项目类型

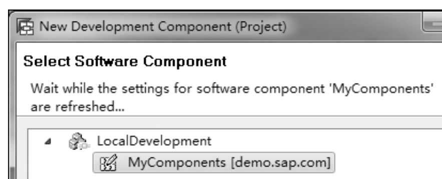


图 7-41　选择本地组件

4）在弹出的对话框中输入工程名称，并单击“Next”按钮，如图 7-42 所示。

5）配置项目公共部分，使用默认配置即可，单击“Finish”按钮完成项目的创建，如图 7-43 所示。



图 7-42　输入项目信息

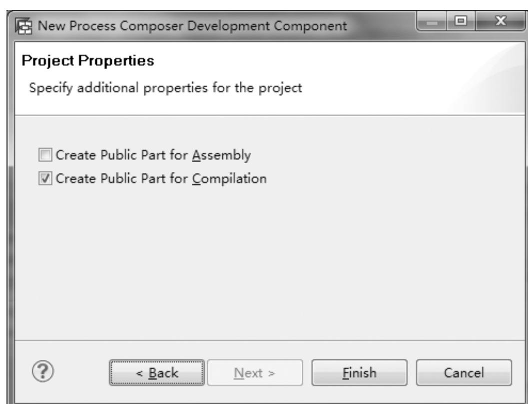


图 7-43　选择创建公共部分

6）创建完成后，就能在项目浏览器中看到整个工程的完整目录结构，如图 7-44 所示。

7）展开“Process Modeling”，随后右击“Processes”，在弹出的快捷菜单中选择“New Process”进行流程的创建，如图 7-45 所示。

8）在弹出的对话框中输入流程的名称“VacationReqPro”，勾选创建一个流程池并输入名称“VacationReqPool”和泳道名称“Dept”，其他使用默认设置，随后单击“Next”按钮，如图 7-46 所示。

9）选择“New”，创建一个新的流程启动触发器，如图 7-47 所示。

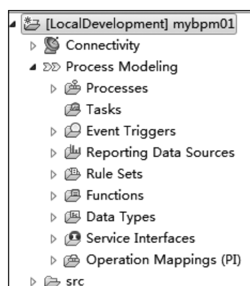


图 7-44　创建后的目录结构

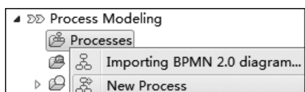


图 7-45　创建流程的菜单选择

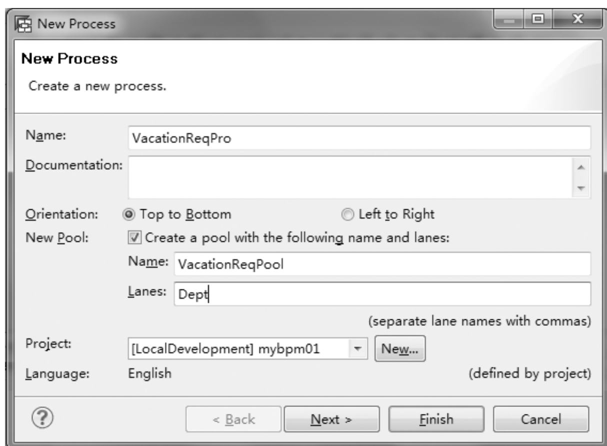


图 7-46　创建新流程

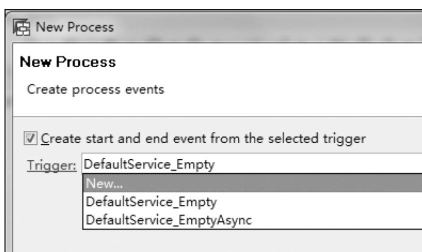


图 7-47　流程启动触发器

10）在弹出的对话框中输入触发器的名称“StMes”，随后单击“Next”按钮，如图 7-48 所示。

11）服务接口选择默认接口，随后单击“Finish”按钮完成触发器的创建，如图 7-49 所示。

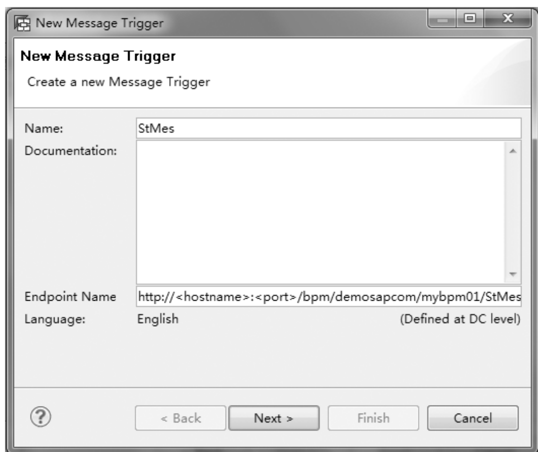


图 7-48　输入触发器信息

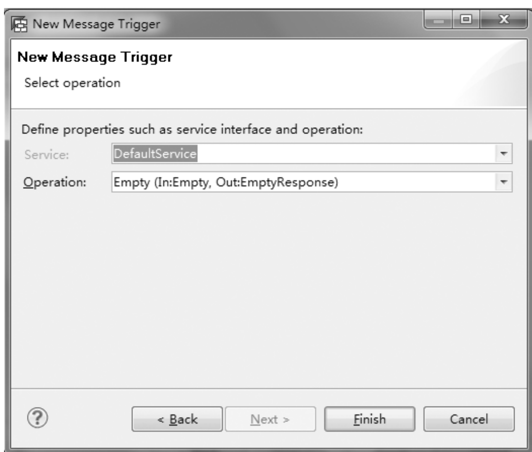
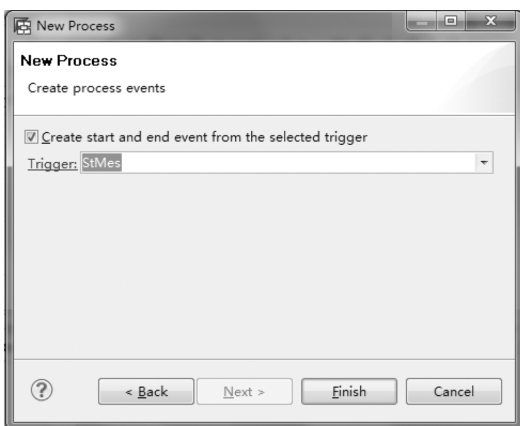


图 7-49　输出触发器运行事件

12）单击“Finish”按钮，完成流程的创建，如图 7-50 所示。

13）在右侧的模型编辑器中可以看到创建完毕的休假申请空流程，如图 7-51 所示。



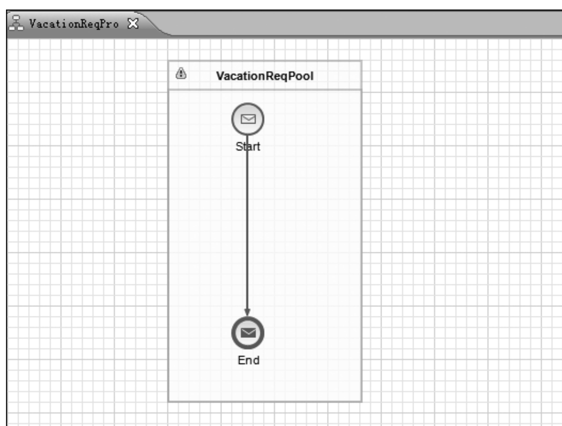
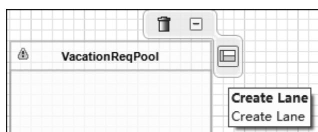


图 7-50　选择触发器　　　　图 7-51　流程建模

14）将鼠标悬停在标题上，单击弹出的“创建泳道”，如图 7-52 所示。

15）输入泳道名称“Com”代表公司审批阶段，如图 7-53 所示。



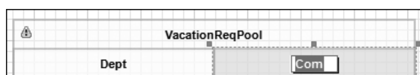
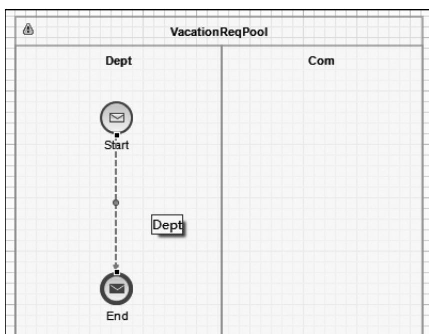


图 7-52　弹出菜单　　　　图 7-53　创建新泳道

16）选中流程中的“Start”至“End”的连线（选中有高亮显示），删除，如图 7-54 所示。

17）将鼠标悬停在开始事件上，并选择创建一个用户动作，如图 7-55 所示。



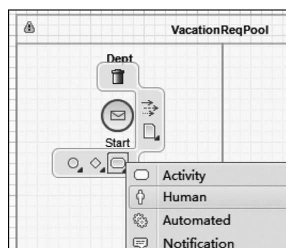


图 7-54　建模操作　　　　图 7-55　创建用户动作

18）完成动作创建后，编辑器中将出现对应的图形，如图 7-56 所示。

19）接着将鼠标悬停在用户动作结点并添加网关，用于判断流程走向。选择“Exclusive Choice”，如图 7-57 所示。

20）接着将鼠标悬停在网关上，继续创建第二级审批的用户动作，并将对象拖曳至公司审批的泳道中。虽然泳道并没有对代码或配置产生任何影响，但是明细的泳道划分有利于后期的修改及运维管理，如图 7-58 所示。

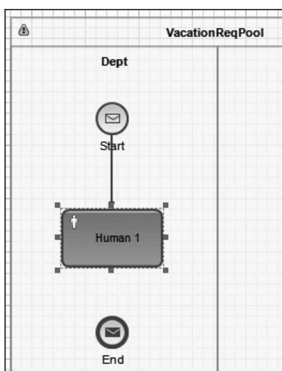


图 7-56　创建用户动作

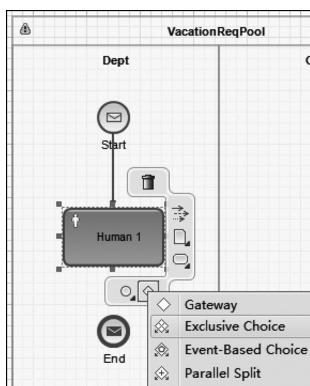


图 7-57　创建网关的菜单选择

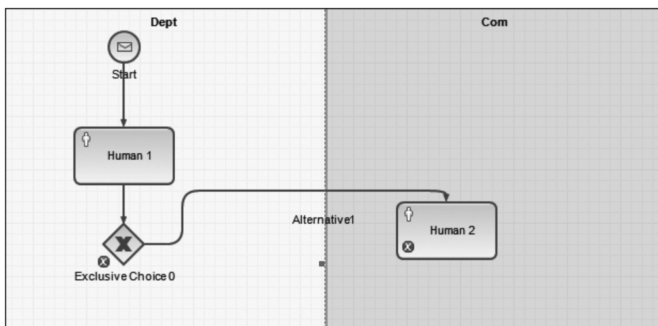


图 7-58　创建新的用户动作

21）接着在二级审批的基础上继续创建网关“Exclusive Choice 1”，随后再创建一个发起人处理结点“Human3”，如图 7-59 所示。

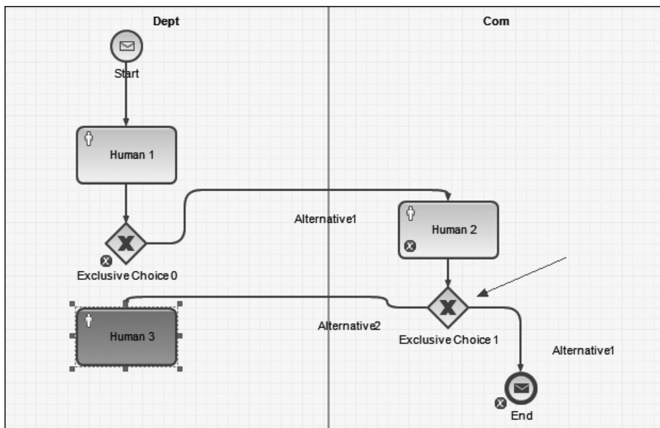


图 7-59　建模操作

22）在“Human3”结点基础上，单击连线并拖曳至“Human1”结点，建立发起人结点到一级审批的联系，如图 7-60 所示。

23）整个流程模型构建完毕，进一步修改结点描述。通过流程图可以看出，该流程为二级审批，申请人提交后直接进入一级审批，一级审批通过则到二级审批，二级审批通过流程结束，一级和二级拒绝都会驳回到申请人，申请人重新提交并再次到一级审批，如图 7-61 所示。

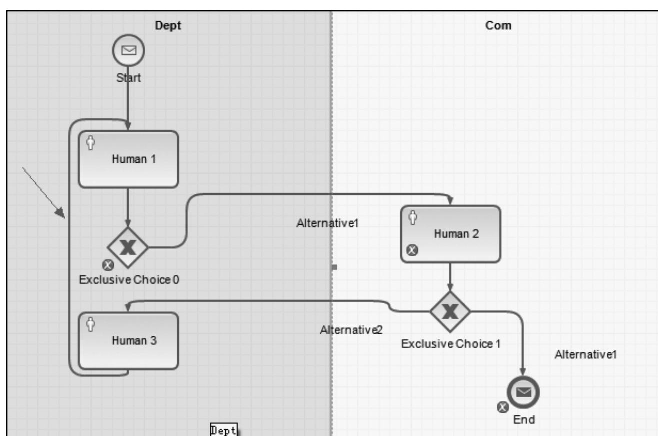


图 7-60　创建联系

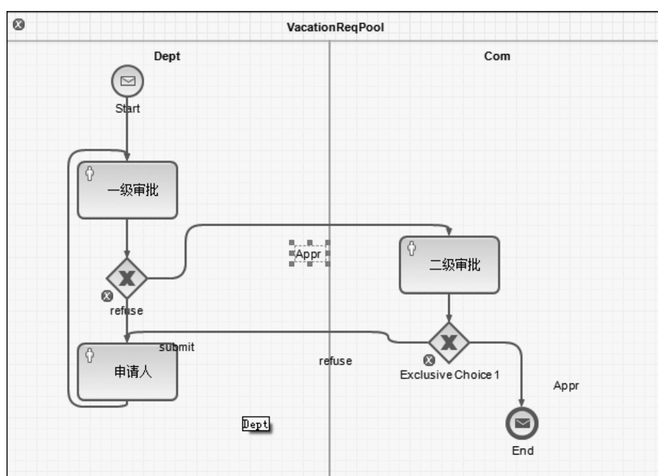


图 7-61　流程示例

24）接着创建流程中的任务，右击“Tasks”文件夹，选择“NewTask”，创建一个新的任务，如图 7-62 所示。

25）输入任务名称，随后单击“Finish”按钮，如图 7-63 所示。

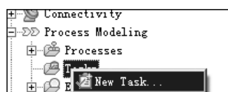


图 7-62　创建任务的菜单选择

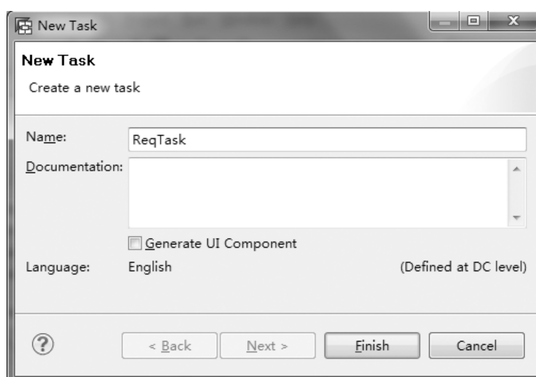


图 7-63　输入任务信息

26）按照同样的操作，分别建立 DeptTask、ComTask 和 ReqTask，3 个任务分别对应流程中的 3 个用户动作，如图 7-64 所示。

27）接着选中第一个用户动作“一级审批”，在下方属性的页签中，找到“Task”子标签，在其中选择对应的任务“DeptTask”，表示部门审批环境。用同样的方法将“二级审批”的任务选择为“ComTask”，将“申请人”的任务选择为“ReqTask”，如图 7-65 所示。

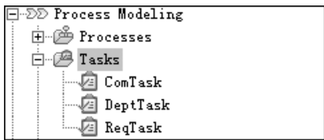


图 7-64　创建后的目录结构

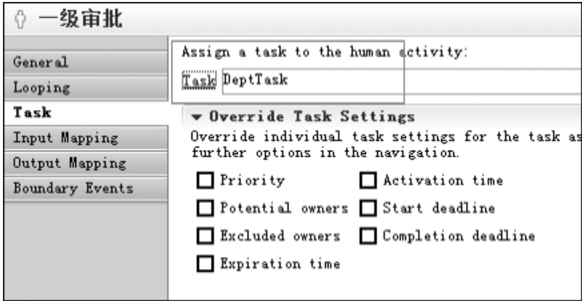


图 7-65　在用户动作中选择对应任务

7.4.2　关联 WD4J 项目

在 Web Dynpro 工程中，只有接口视图和接口控制器是对外可见的。对于 BPM 流程中的任务界面定义，需要进行绑定的就是 Web Dynpro 中的接口视图。

开发实例：

1）首先创建一个 Web Dynpro 工程，选择“File”→“New”→“Project”（见图 7-66），或单击新建项目按钮。

2）在弹出的对话框中选择对应的项目类型，找到“Web Dynpro Development Component”单击“Next”按钮，如图 7-67 所示。

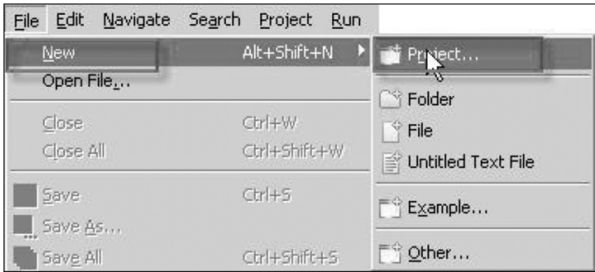


图 7-66　创建项目的菜单选择

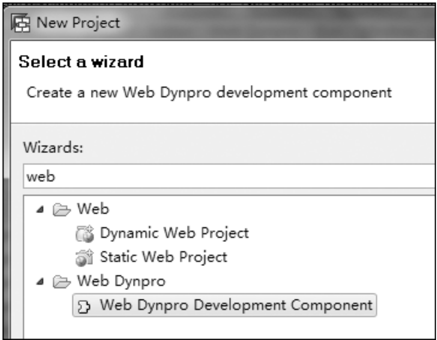


图 7-67　选择项目类型

3）选择本地的开发架构中的“MyComponents”，单击“Next”按钮，如图 7-68 所示。

4）输入工程名称“mydynp08”，单击“Next”按钮，如图 7-69 所示。

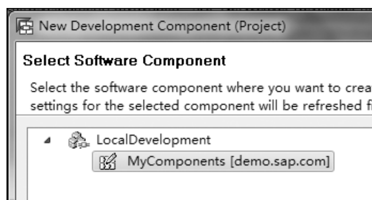


图 7-68　选择本地开发组件



图 7-69　输入项目信息

5）展开创建后的 Web Dynpro 工程，右击“Components”并选择“Create Component”，创建一个组件，如图 7-70 所示。

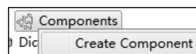


图 7-70　创建组件的菜单选择

6）输入组件名称“Dynp08_01Comp”，单击“Next”按钮，如图 7-71 所示。

7）输入对应的窗口名称“Dynp08_01CompWindow”和接口视图名称“Dynp08_01Comp_01InterV”，单击“Finish”按钮，如图 7-72 所示。

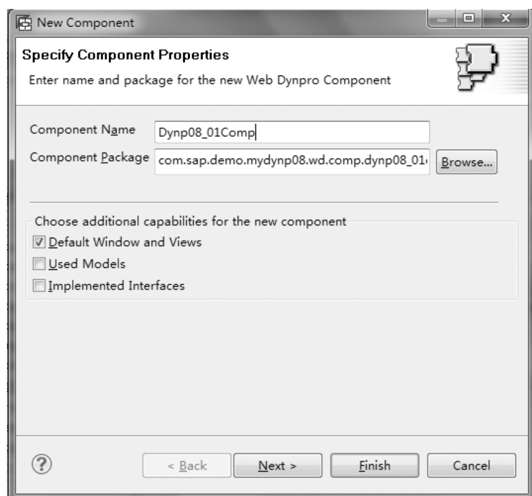


图 7-71　输入组件信息

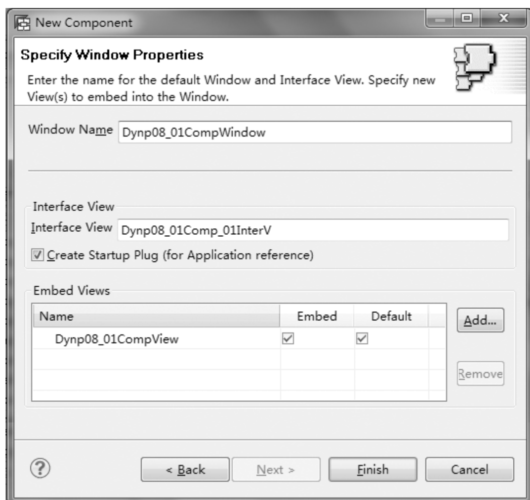


图 7-72　输入窗口和视图信息

8）默认创建的接口视图作为流程发起时申请人操作的对应页面，接着再创建两个接口视图“Dynp08_01Comp_02InterV”和“Dynp08_01Comp_03InterV”做为一级审批与二级审批的任务页面，如图 7-73 所示。

9）在新建的接口视图中，需手动创建一个类型为“Startup”的 Inbound Plug 作为默认入口，如图 7-74 所示。

10）在创建两个接口视图后，还需要创建两个窗口“Dynp08_02CompWindow”和“Dynp08_03CompWindow”，用于实现新建的接口视图，如图 7-75 所示。接口视图要与窗口一一对应。

11）同时要保证 3 个窗口的“接口视图”属性中都已实现了对应的接口视图，如图 7-76 所示。

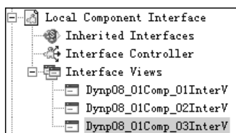


图 7-73　创建后的目录结构

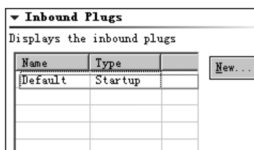


图 7-74　创建的入口

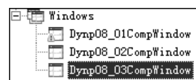


图 7-75　创建对应窗口

12）编辑申请人操作的视图，创建上下文结点及相关属性，如图 7-77 所示。

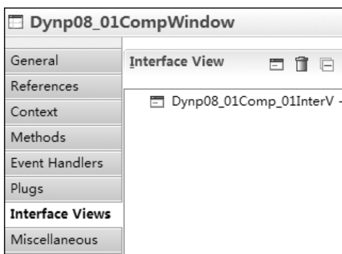


图 7-76　实现对应接口视图

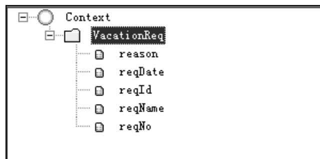


图 7-77　上下文清单

13）通过表单的模板向导自动生成请假申请的页面，并添加提交按钮，如图 7-78 所示。

14）使用相同的上下文以及视图元素创建另外两个视图，但在审批页面中需添加“通过”与“拒绝”两个按钮用于触发 BPM 流程的流转，同时页面元素属性均修改为只读，如图 7-79 所示。

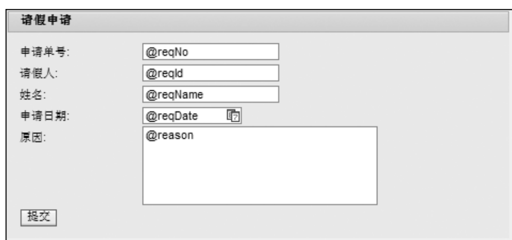


图 7-78　预览效果



图 7-79　运行效果

需要注意的是，由于现在是基于 BPM 进行开发，每个 Web Dynpro 的视图都是被 BPM 流程中一个独立的任务所调用并展示，通过 BPM 的流程分别将不同的待办任务推送给用户，因此并不存在 Web Dynpro 中的页面切换动作，之前页面的元素中绑定的上下文也就不需要与组件控制器中的上下文做映射。

15）接着创建组件控制器中的上下文，这个上下文是用于提供给 BPM 流程进行**流程上下文映射、网关判断逻辑、任务审批人、任务标题**等内容的配置。创建存放数据的对应结点：ProcessNode、BusiData、Reserv1Node、Reserv2Node 和 Reserv3Node，如图 7-80 所示。其中“ProcessNode”的集合属性是“1··1”，因为每次用户仅提交一份表单。

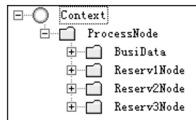


图 7-80　上下文结点清单

16）其中“BusiData”包含了具体的业务数据特性，集合属性是“1··1”，如图 7-81 所示。

17）其余 3 个结点用于作为预留字段，避免接口频繁更改带来的麻烦，集合属性是“1

· · n”，如图 7-82 所示。

18）接着右击并复制组件控制器中的上下文，打开 Web Dynpro 的接口控制器并粘贴，完成上下文结点对外声明的动作，如图 7-83 所示。

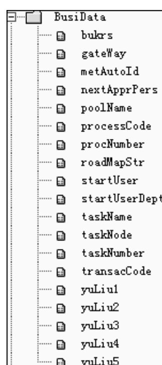


图 7-81　特性清单

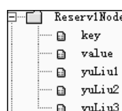


图 7-82　特性清单

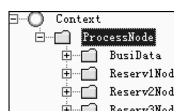


图 7-83　上下文清单

19）在组件控制器中创建一个名称为“finish”的自定义事件（Events），将该事件复制并粘贴至接口控制器中，如图 7-84 所示。

20）创建一个自定义方法用于处理事件，不需要定义参数，如图 7-85 所示。

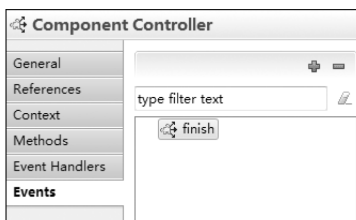


图 7-84　自定义事件

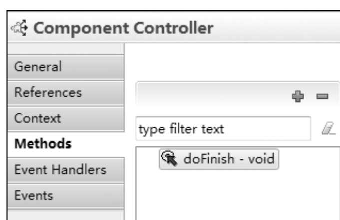


图 7-85　自定义方法

21）保存所有修改后，进入 Java 编辑器，添加事件的触发代码：

```
public void finish( )   {
  //@ @ begin finish( )
  wdThis. wdFireEventFinish( ) ;
  //@ @ end
}
```

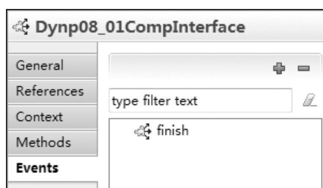


图 7-86　自定义事件

22）将事件复制并粘贴至接口控制器中，如图 7-86 所示。

23）在申请人视图的“提交”按钮的动作中添加代码：

```
//获取流程流水号并根据流水号判断是否在流程中
IPublicDynp08_01Comp wdGetDynp08_01CompController = wdThis. wdGetDynp08_01CompController
( ) ;
String  procNumber  =  wdGetDynp08 _ 01CompController. wdGetContext ( ) . nodeBusiData ( )
. currentBusiDataElement( ). getProcNumber( ) ;
if( procNumber == null || procNumber. length( ) ==0) {
  //启动流程
  //产生流程号
```

```
procNumber = "000001";
//保存单据至数据库
//调用标准 API 发起流程
this. startProcess( wdContext. currentVacationReqElement( ). getReqNo( ), procNumber);
} else{
//直接进入一级审批
  currentBusiDataElement. setTaskName("请假申请一级审批");
wdGetDynp08_01CompController. doFinish( );
}
```

24）在审批人视图的“提交”按钮的动作中添加代码：

```
IPublicDynp08_01Comp wdGetDynp08_01CompController = wdThis. wdGetDynp08_01CompController
( );
IBusiDataElement currentBusiDataElement = wdGetDynp08_01CompController. wdGetContext ( )
. nodeBusiData( ). currentBusiDataElement( );
String taskNode = currentBusiDataElement. getTaskNode( );
if("20". equalsIgnoreCase(taskNode)) {
//一级通过
currentBusiDataElement. setTaskNode("30");//设置下一结点为 30
currentBusiDataElement. setTaskName("请假申请二级审批");
} else if("30". equalsIgnoreCase(taskNode)) {
//流程结束
}
//触发监听 event
wdGetDynp08_01CompController. doFinish( );
```

25）在审批人视图的“拒绝”按钮的动作中添加代码：

```
IPublicDynp08_01Comp wdGetDynp08_01CompController = wdThis. wdGetDynp08_01CompController
( );
IBusiDataElement currentBusiDataElement = wdGetDynp08_01CompController. wdGetContext ( )
. nodeBusiData( ). currentBusiDataElement( );
String taskNode = currentBusiDataElement. getTaskNode( );
if("20". equalsIgnoreCase(taskNode)) {
//一级拒绝
currentBusiDataElement. setTaskName("请假申请一级驳回");
} else if("30". equalsIgnoreCase(taskNode)) {
//二级拒绝
currentBusiDataElement. setTaskName("请假申请二级驳回");
}
currentBusiDataElement. setTaskNode("10");//设置下一结点为 10
//触发监听 event
wdGetDynp08_01CompController. doFinish( );
```

26）右击“Dynp08_01Comp”，在弹出的快捷菜单中选择“Add to Public Part”，如图 7-87 所示。

27）在弹出的对话框中单击“New”按钮，创建一个公共部分，输入名称“api”，随后单击“Finish”按钮，如图 7-88 所示。

28）完成公共部分创建后，单击“Finish”按钮，如图 7-89 所示。

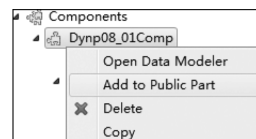


图 7-87　添加公共部分的菜单选择

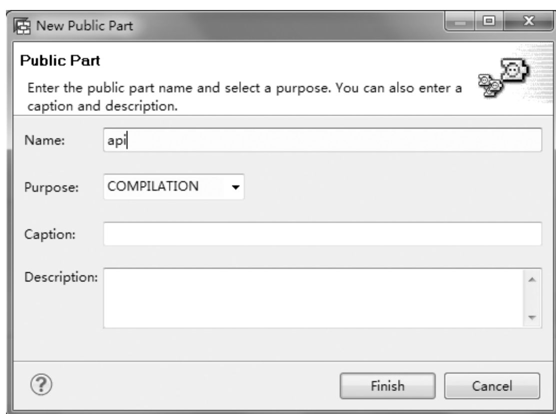


图 7-88　输入公共部分信息

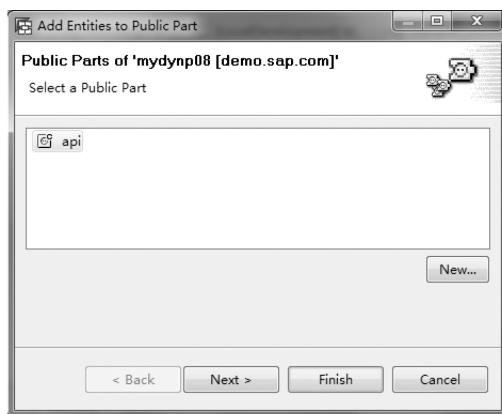


图 7-89　选择公共部分

29）通过开发工具的视图切换按钮，切换至“Development Infrastructure”，如图 7-90 所示。

30）在左侧的“MyComponents”中选择 BPM 工程“mybpm01”，在右侧属性中打开“Dependencies”依赖标签，单击“Add”按钮添加依赖，如图 7-91 所示。

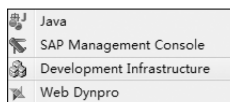


图 7-90　切换透视图



图 7-91　添加项目依赖

31）添加后，保存所有修改，并编译两个工程，如图 7-92 所示。

32）在完成 Web Dynpro 工程的开发与配置后，重新进入 BPM 流程编辑视图，打开流程中的任务，切换到“Overview”，如图 7-93 所示。

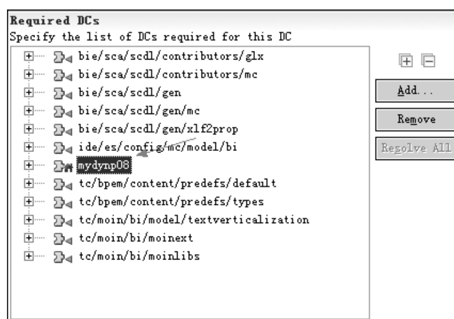


图 7-92　添加项目依赖

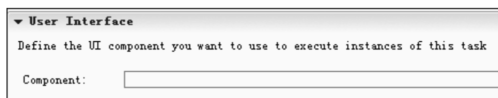


图 7-93　绑定 Web Dynpro 工程界面

33）单击“User Interface”，选中组件右侧的“Choose”单选按钮，在弹出的对话框选择默认选项，随后单击“OK”按钮，如图 7-94 所示。

34）单击“Search”按钮，搜索可用的 Web Dynpro 工程公共部分，选择对应的接口视图，随后单击“Next”按钮，如图 7-95 所示。

35）选择视图按钮对应触发的事件，将 BPM 中的流程流转动作与 Web Dynpro 中的事件绑定，如图 7-96 所示。

36）完成 BPM 中任务与 Web Dynpro 接口视图的绑定，如图 7-97 所示。

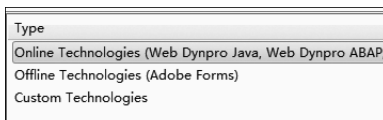


图 7-94　选择对应类型

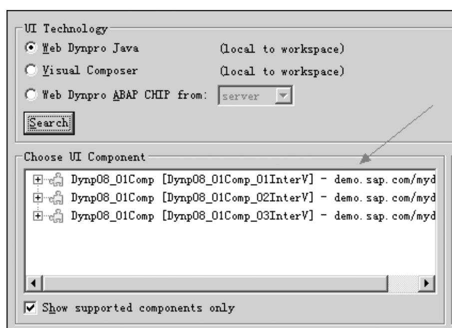


图 7-95　选择对应组件

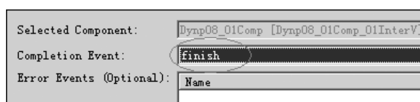


图 7-96　选择对应事件

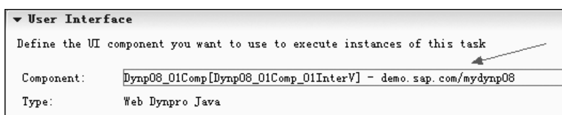


图 7-97　配置完成后的界面

7.4.3　映射流程上下文

1）在上一节中，完成了 BPM 中的流程流转动作与 Web Dynpro 中的事件绑定。而同时在 BPM 工程的数据类型中也完成了对 Web Dynpro 上下文的引用，展开 BPM 工程目录中的“Data Types”就能看到引用的结果，如图 7-98 所示。

2）将上下文的根结点结构“Context”拖曳至右侧模型编辑器中，生成了“DO_Context”的流程上下文对象，即完成了流程上下文的声明。完成上下文定义后，需要完成**流程上下文映射、网关判断逻辑、任务审批人、任务标题**的配置，如图 7-99 所示。

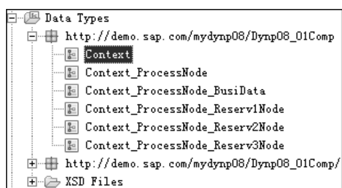


图 7-98　绑定 Web Dynpro 项目后的效果

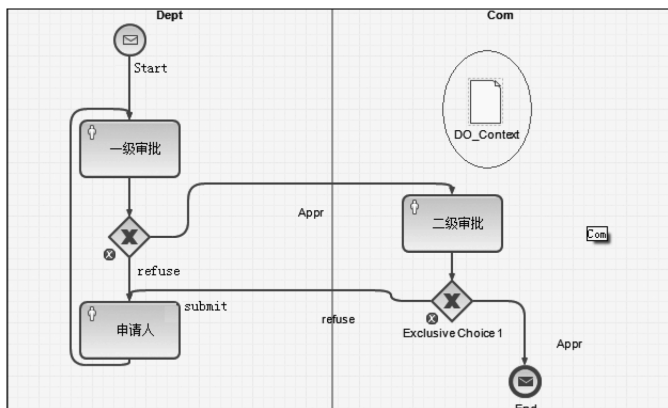


图 7-99　生成“DO_Context”对象

3）接着选中用户动作，打开下方的“Input”页签，完成从左侧流程上下文到右侧流程任务所需上下文的映射，如图 7-100 所示。

4）打开下方的“Output”页签，完成从左侧流程任务输出的上下文到右侧流程上下文的映射，如图 7-101 所示。

5）按照同样的操作，将其余的用户动作结点进行上下文映射。参考 Web Dynpro 开发模式，可以将“DO_Context”流程上下文对象想象成 Web Dynpro 中的组件控制器，工作流中的数据流转都需要通过“DO_Context”进行同步，但与组件控制器不同的是，“DO_Context”

只有数据定义而没有逻辑定义。

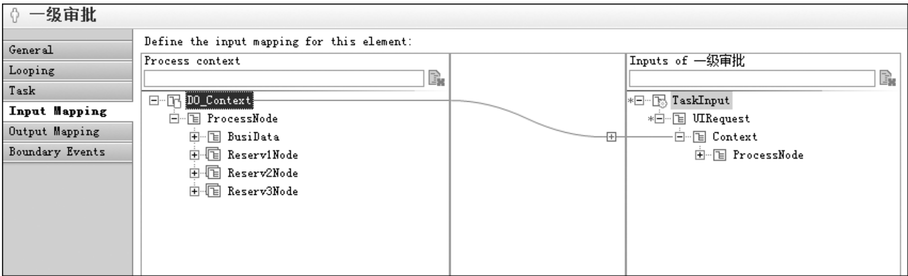


图 7-100　流程上下文映射 1

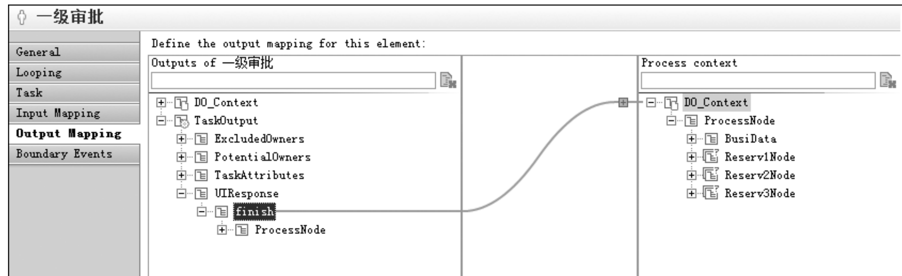


图 7-101　流程上下文映射 2

6）接着进行网关判断的定义。选中流程中的网关，勾选“refuse”为默认的，接着选择“Appr”并单击“Edit”按钮，打开条件编辑，如图 7-102 所示。

Define the condition evaluation order for this gateway. Condition for default gate is not evaluated.

O..	Name	Condition	Default Gate
1	Appr	<no condition>	☐
2	refuse	true	☑

Up　Down　Edit

图 7-102　修改网关判断条件

7）根据设计，当上下文中的“tasknode”特性值为“30”时，代表审核通过并到二级审批步骤，默认为拒绝。条件表达式的关键字可以从右侧的清单中选择，也可以手动编写，如图 7-103 所示。输入对应条件表达式：

```
string - equal(DO_Context/ProcessNode/BusiData/taskNode,"30")
```

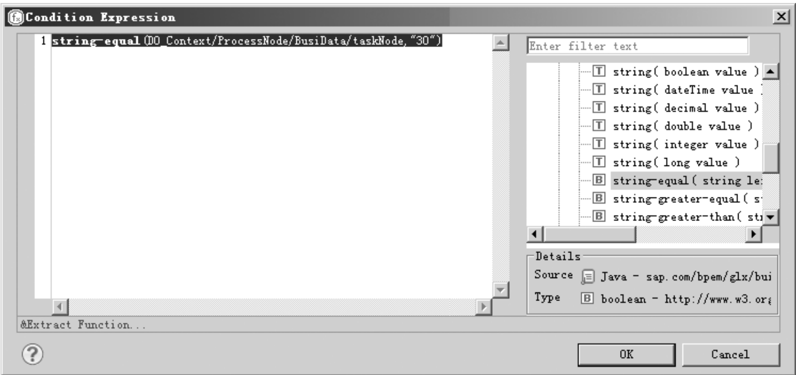


图 7-103　修改网关判断条件表达式

8）使用同样的方式设置二级审批的网关判断条件。

9）接着选中任务，切换至“Roles”角色页签，选中“Use an expression”单选按钮，随后单击“Edit”按钮，如图 7-104 所示。

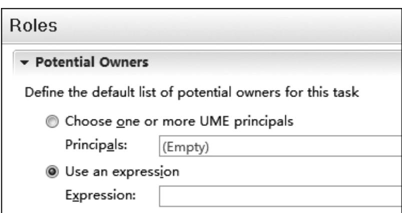


图 7-104　编辑角色所有者

10）表达式的关键字与网关逻辑判断时一样，审批人通过上下文中的“nextApprPers”传递，如图 7-105 所示。

```
getPrincipal(UIRequest/Context/ProcessNode/BusiData/nextApprPers)
```

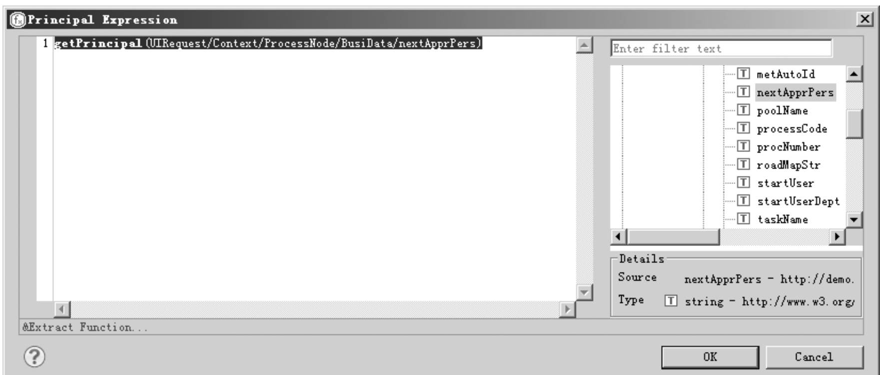


图 7-105　编辑表达式

11）使用同样的方式设置另外两个流程任务的审批人。

12）接着打开任务的“User Texts”页签，在“Variables”中单击“Add”按钮添加一个“taskName”的参数，随后单击“Edit”按钮，如图 7-106 所示。

▼ Variables

Define the variables you want to use to parameterize the user texts below

Variable:

Name	Type	Expression
taskName	string	No expression defined

图 7-106　添加变量

13）使用上下文中的“taskName”进行传递（见如图 7-107），输入：

```
UIRequest/Context/ProcessNode/BusiData/taskName
```

14）完成变量创建后，接着在下方的任务标题中输入该参数，在“Subject”中输入“{taskName}”，其中“{}”表示引用参数的名称。完成设置后，所有待办的标题也可以通过 Web Dynpro 中的上下文进行自定义修改了，如图 7-108 所示。

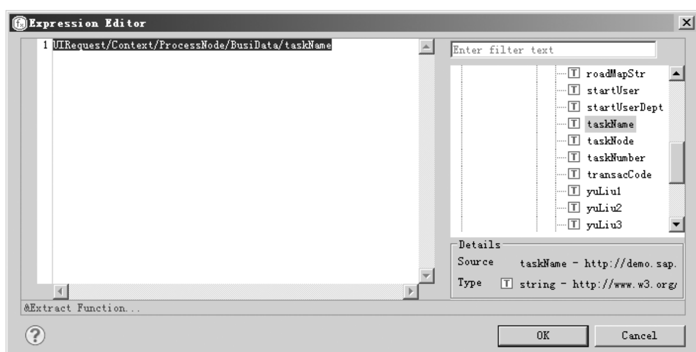


图 7-107　编辑表达式

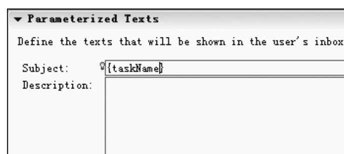


图 7-108　定义任务标题

7.4.4　配置启动方式

1）在“Service Interface”服务接口中创建流程启动的 Web 服务，右击“WSDL Files”文件夹，选择“New WSDL”，如图 7-109 所示。

2）输入 WSDL 文件的名称“StartInter”，随后单击“Finish”按钮，如图 7-110 所示。



图 7-109　创建新的 WSDL 文件

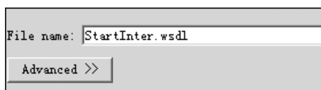


图 7-110　输入文件名

3）创建完成后，编辑其中的参数。由于 BPM 的 Web 服务是异步的服务，不需要即时响应的输出，首先删除“Output”返回结构，如图 7-111 所示。

4）单击“Source”源代码页签，输入其中与“Response”相关的代码，如图 7-112 所示。

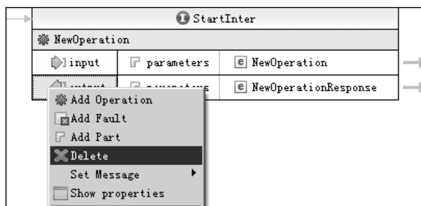


图 7-111　编辑参数

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.example.org/StartInter/">
      <xsd:element name="NewOperation">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="NewOperationRequest">
    <wsdl:part element="tns:NewOperation" name="parameters"/>
  </wsdl:message>

  <wsdl:portType name="StartInter">
    <wsdl:operation name="NewOperation">
      <wsdl:input message="tns:NewOperationRequest"/>

    </wsdl:operation>
```

图 7-112　添加代码

5）接着修改 WSDL 文件的输入参数结构，将 Web Dynpro 生成的“xsd”上下文定义文件重命名，并导入到 BPM 工程的“XSD Files”中，如图 7-113 所示。

6）单击方法右侧的箭头，进入方法参数配置的页面，如图 7-114 所示。

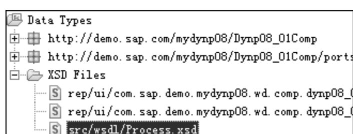


图 7-113　导入 xsd 文件



图 7-114　编辑执行方法

7）删除默认参数，右击“NewOperation”，选择设置类型，如图 7-115 所示。

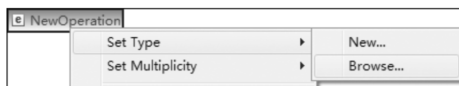


图 7-115　编辑执行方法参数

8）在弹出的对话框中找到导入的 Web Dynpro 上下文结构的 XSD 文件，完成导入，如图 7-116 所示。

9）在目录中选择之前创建的“StMes”触发器，在右侧找到设置属性并选择服务接口“StartInter”，如图 7-117 所示。

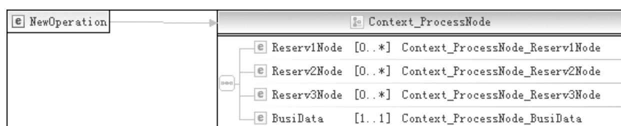


图 7-116　完成参数创建后的效果

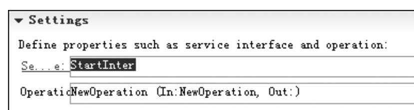


图 7-117　触发器配置

10）随后选中开始事件，打开下方的“Output”页签，将左侧开始事件的传入参数与右侧流程上下文进行映射，即表示在流程启动后随着 Web 服务传入的数据进入到了流程上下文中，如图 7-118 所示。

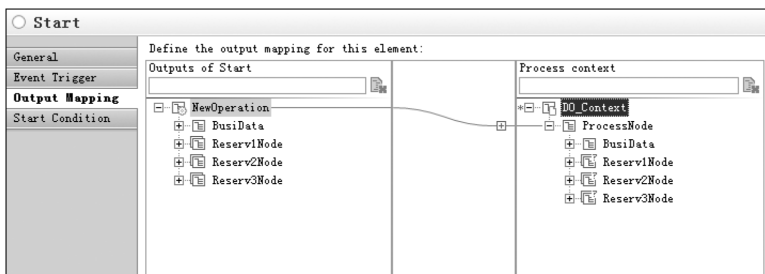


图 7-118　流程上下文映射

至此完成了 BPM 的设计开发过程。

7.4.5　运行测试流程

1）在完成所有的开发配置工作后，分别编译并部署 Web Dynpro 工程和 BPM 工程。部署成功之后，进入服务器端的 NWA 管理员平台，在菜单目录中依次找到“Configuration”→“Processes And Tasks”→“Process Repository”流程资源库。如果之前的部署没有问题，在流程资源库中，就能看到对应的“mybpm01”流程，如图 7-119 所示。

Component (Vendor)	Last Installed Version
mybpm01 (demo.sap.com)	20131218153350 (localDevelopment)

图 7-119　流程资源库

2）在企业门户中，可以看到 Web Dynpro 程序的默认发起流程页面，输入信息并单击“提交”按钮，启动流程，如图 7-120 所示。

3）使用一级审核人员用户登录系统，打开通用工作清单，查看待办任务，如图 7-121 所示。

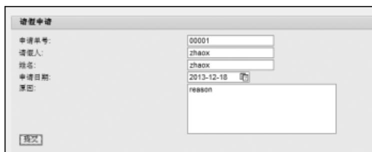


图 7-120　运行效果

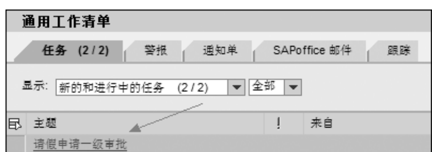


图 7-121　通用工作清单界面

4）单击任务标题链接后，就会打开对应的 BPM 处理页面，标题也是通过 Web Dynpro 中的代码动态生成的“请假申请一级审批”，如图 7-122 所示。

5）单击“通过”按钮，完成第一级审批，如图 7-123 所示。



图 7-122　待办任务处理界面

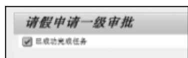


图 7-123　审批后提醒消息

6）接着使用二级审核人员用户登录系统，打开通用工作清单，查看待办任务，如图 7-124 所示。

7）单击任务标题链接，可以进行二级审核，如图 7-125 所示。

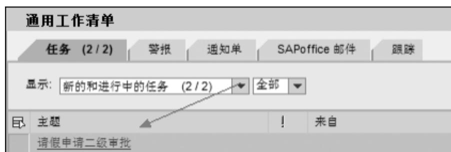


图 7-124　通用工作清单界面



图 7-125　二级审批界面

7.4.6　流程监控

1）使用管理员账号登录 NWA 平台，在菜单中选择“Configuration”，随后单击“Processes and Tasks”链接就能打开流程和任务的管理界面，在页面中可以查看流程的历史版本以及流程流转信息，如图 7-126 所示。

2）除了监控功能外，管理员还可以对流程做特殊处理，如图 7-127 所示。

3）单击“Manage Processes”链接，能够监控到新发起的流程，如图 7-128 所示。

4）通过“Details of Process Instance”可以查看流程的上下文和流程定义，如图 7-129 所示。

Details of Component mybpm01 (demo.sap.com) and Version 20131219005942 (localDevelopment)

Processes and Tasks | Resources

Process Flow | Start Process...

Name
ComTask
DeptTask
ReqTask
VacationReqPro

图 7-126　流程资源库中的任务清单

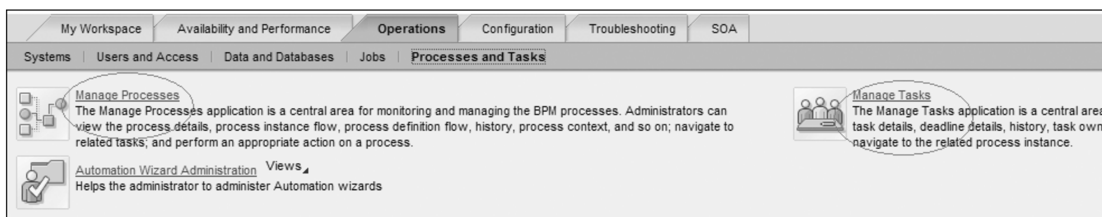


图 7-127　NWA 平台中的流程管理和任务管理

Show: All Running Processes | Actions | Show Process Flow | Show Related Tasks | Archive... | Recover | Export Process Detail

Find Process Instance: Go

Error Status	Status	Process Name	Process Subject	Process Instance ID
OK	In Progress	VacationReqPro	Pool 0	eca993cd684811e3ac0400000048cc86

图 7-128　查看流程状态

Details of the Process Instance VacationReqPro

Details | Process Definition | Administrators | History | Context Data | Error Log

Show: DO_Context | Edit | Save and Resume | Cancel

Name	Value
DO_Context	
ProcessNode	
BusiData	
startUserDept	0001
yuLiu3	

图 7-129　查看流程中的上下文

5）选中监控的流程，单击“Show Process Flow”按钮，如图 7-130 所示。

Manage Processes: Process Instances

Favorites | Related Links | Go To | Support Details

Show: All Running Processes | Actions | Show Process Flow | Show Related Tasks | Archive... | Recover | Export Process Detail

Find Process Instance: Go

Error Status	Status	Process Name	Process Subject	Process Instance ID
OK	In Progress	VacationReqPro	Pool 0	eca993cd684811e3ac0400000048cc86

图 7-130　查看流程流转图

6）可以查看运行时的流程流转图，根据上面流程的运转情况可知该流程处于二级审批，如图 7-131 所示。

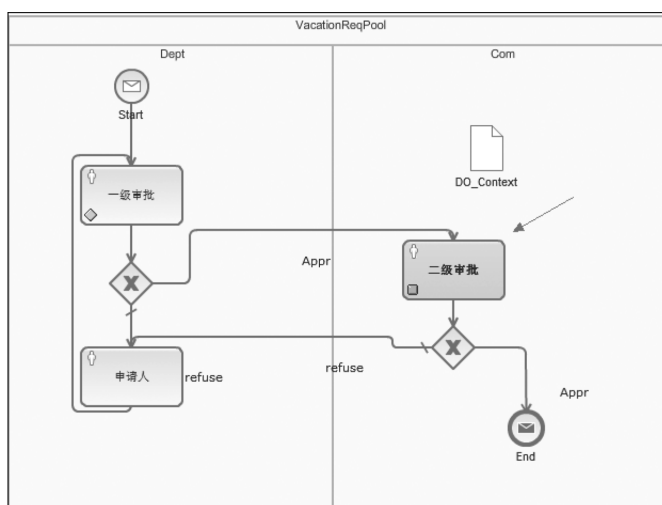


图 7-131　标准流程流转图

7）单击“Show Related Tasks”，可以查看当前流程实例的待办处理状态，同时可以监控指定任务的上下文中的变量值，如图 7-132 所示。

List of Task Instances

Show: <Custom View> | Actions | Show Process | Nominate Owner... | R

Find Task Instance: Go

Status	Lifecycle Status	Task Definition	Task Instance ID
In Progress	In Progress	ComTask	45a02bb2684911e3c1ba…
Completed	Completed	DeptTask	ecc8804f684811e3857c0…

图 7-132　相关任务实例清单

第 8 章　NWA 相关技术

对于 SAP NetWeaver 平台的开发来说，Web Dynpro 是最主要的技术，但随着平台不断的升级和功能扩展，也逐步兼容和集成了很多相关的成熟技术；在开发技术拓展的同时，NetWeaver 平台也对相应技术提供了后台保障。简单来说，代码层面的开发交给 Web Dynpro，而配置、运行和管理等工作就交给了 NWA，这对于项目开发实施的稳定性以及用户体验的提升都起了非常大的作用。本章将对 NWA 管理者平台、服务器实例与应用、作业与定时任务、Adobe 技术交互表单、自定义数据源等进行介绍，并提供相关开发实例供大家参考。

8.1　NWA 管理者平台

从 SAP NetWeaver 平台的角度来看，SAP 提供了若干管理工具，其中 SAP NetWeaver 管理者平台（SAP NetWeaver Administrator，NWA）主要用于在线的配置修改和管理，而 Config tool 则用于一些离线的配置管理（大部分修改后需要重启服务器生效）。在 SAP NetWeaver 7.0 以及之前的版本中，系统管理员主要通过 Visual Administrator（简称 VA）来做系统配置管理，但在新版本的平台中，VA 管理工具已集成至（或升级成为）NWA。

SAP NetWeaver 管理者平台（NWA）是一个基于网页端的，可提供管理员进行管理、配置以及监控服务器端内容的框架工具。它由若干独立的管理工具组成，可以用于管理、配置及监控 SAP NetWeaver 系统、它的组件以及所有运行的应用程序。SAP NetWeaver 管理者平台具有以下几个特点：

1）它是处理管理员任务的一个核心工具，如启动或停止一个服务器实例、修改配置设定、分析系统日志、分析报错信息并且确保所有系统组件都能正常运行。

2）它的界面分类面向不同的管理员任务，易于上手使用。

3）它完全基于网络运行，不需要在本地进行额外的安装和卸载工作。

4）在不同界面提供了无缝的链接导航，方便管理员进行不同操作任务的跳转。

5）在个别处理界面中，将多个专家工具集成在一起，形成一个简单、清晰并可以集中处理一系列问题的页面，方便管理员使用。

如需要访问 SAP NetWeaver 管理者平台，首先要确保服务器正常运行，随后在浏览器中输入下列地址：http://<host>:<port>/nwa。其中，“host”是 SAP NetWeaver Java 应用服务器的访问域名，“port”是对应的 HTTP 端口，默认是“5<Java_instance_number>00”，一般在开发机中比较常见的端口即“50000”。

当然访问的用户只有拥有必要的角色权限，才能使用管理者平台。在对应的用户管理中可以找到“NWA_READONLY”和“NWA_SUPERADMIN”两个标准角色，分别对应 NWA 的只读权限以及 NWA 的超级管理员权限（默认管理员账户 administrator 已拥有了上述角色权限，可以直接登录并使用）。

在 NWA 中，所有核心的管理员功能都被一一分类并放入不同的 NWA 管理中心，包括

"Operation"运行工作中心、"Configuration"配置工作中心、"Availability and Performance"可用及性能工作中心、"Troubleshooting"问题处理工作中心和"SOA"面向服务工作中心等。

单击"Site Map"链接，可以看到所有工作中心的具体功能服务清单，如图 8-1 所示。

我的工作空间
收藏夹
最近使用
配置向导 (/nwa/cfg-wizard)
应用程序模块 (/nwa/app-modules)
BPM 系统概览
系统概览
流程资源库 (/nwa/bpm-repository)
日志查看器 (/nwa/logs)
目标模板管理 (/nwa/destinationtemplates)
Java 调度器 (/nwa/scheduler)
管理任务 (/nwa/bpm-tasks)
管理流程 (/nwa/bpm-processes)
启动和停止 (/nwa/start-stop)
应用程序资源 (/nwa/app-resources)
Java 系统属性 (/nwa/sys-config)

可用性和性能
系统概览
系统概览
资源监控
JCo 监控 (/nwa/connection-mon)
锁 (/nwa/locks)
分布式事务
系统性能统计
会话管理 (/nwa/sessionmgt)
历史记录报表
流程监控
消息监控器
BPM 系统概览
BPM 系统概览

操作
系统
应用程序管理器
启动和停止 (/nwa/start-stop)
用户和访问
系统计量
数据和数据库
ILM 存储浏览器 (/nwa/ilm-storebrw)
Java 归档主控室 (/nwa/archcockpit)

配置
安全性
验证和单点登录 (/nwa/auth)
目标 (/nwa/destinations)
身份管理 (/nwa/identity)
证书和密钥 (/nwa/key-storage)
受信系统 (/nwa/trusted-systems)
病毒扫描提供者 (/nwa/virus-scan)
SSL (/nwa/ssl)
ClickJacking Whitelist Configuration
基础架构
Adobe 文档服务 (/nwa/adobe)
应用程序模块 (/nwa/app-modules)
应用程序资源 (/nwa/app-resources)
Java 配置浏览器 (/nwa/cfg-browser)
目标 (/nwa/destinations)
Java HTTP 提供者配置 (/nwa/http)
国际化 (/nwa/internationalization)
Java 类加载器查看器 (/nwa/classloader)
JCo RFC 提供者 (/nwa/jco-destinations)
JMS 服务器配置 (/nwa/jms)
许可证 (/nwa/licenses)
日志配置 (/nwa/log-config)
消息服务器 (/nwa/msg-server)
服务注册表管理 (/nwa/SRConfig)
系统架构目录数据提供者配置 (/nwa/sld-config)
系统信息 (/nwa/sysinfo)
Java 系统属性 (/nwa/sys-config)
受信系统 (/nwa/trusted-systems)
开发基础架构
货币管理
计量单位管理
企业内容管理
场景
配置向导 (/nwa/cfg-wizard)
进程和任务
流程资源库 (/nwa/bpm-repository)
连接性
目标 (/nwa/destinations)
Java HTTP 提供者配置 (/nwa/http)
JCo RFC 提供者 (/nwa/jco-destinations)

故障排除
Java
Java 类加载器查看器 (/nwa/classloader)
JNDI 浏览器 (/nwa/jndi)
锁 (/nwa/locks)
消息服务器 (/nwa/msg-server)
系统信息 (/nwa/sysinfo)
比较系统
分布式事务
系统性能统计
数据库
JPA 监控器 (/nwa/jpa-mon)
打开 SQL 数据浏览器 (/nwa/databrowser)
打开 SQL 监控器 (/nwa/sql)
日志和跟踪
日志配置 (/nwa/log-config)
日志查看器 (/nwa/logs)
安全故障排除向导
进程和任务
业务日志 (/nwa/bpm-bizlog)
规则业务日志 (/nwa/brm-bizlog)
流程故障排除 (/nwa/bpm-troubleshooting)
BPM 操作 (/nwa/bpm-actions)
高级故障排除
线程转储分析
堆转储分析

SOA
技术配置
目标模板管理 (/nwa/destinationtemplates)
SOA 中间件全局设置 (/nwa/middleware-settings)
目标 (/nwa/destinations)
服务注册表管理 (/nwa/SRConfig)
JCo RFC 提供者 (/nwa/jco-destinations)
系统连接 (/nwa/systems)
应用程序和场景通信
单个服务管理 (/nwa/ssadmin)
发布规则 (/nwa/publicationrules)
应用程序通信 (/nwa/appcommunication)
用户账户管理 (/nwa/useraccmngt)
业务场景通信 (/nwa/businesscfg)

图 8-1　NWA 站点地图

- 可用性和性能工作中心：提供给管理员监控服务器不同的参数和不同服务的运行状况，如 Java 应用服务器状态、BPM 系统状态、连接器状态以及锁等信息。
- 运行工作中心：提供管理员对服务器进行操作，如启动或停止不同的应用程序，用户权限管理，Java 调度器的配置和使用，Java 服务器实例与应用的管理，BPM 流程和任务的管理等。
- 配置工作中心：用于进行所有与服务器相关的配置工作，如 Adobe 技术交互表单服务、应用程序模块、应用程序资源、系统单点登录、基础配置向导等。
- 问题处理工作中心：提供了系统监控相关内容，如 BPM 运行日志、系统比较、核心服务应用情况、日志查看器等。
- 面向服务工作中心：包含了应用通信、业务场景通信、连接性日志记录、目标管理等。

8.2　服务器实例与应用

启动和停止"Start&Stop"，用于管理服务器实例（instances）的运行状态，同时也可以查看某个指定实例的详细信息，所有的相关 Java 服务以及应用程序也会详细地列在页面中，如图 8-2 所示。

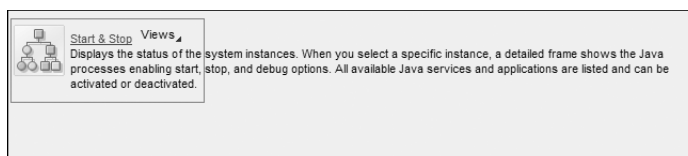


图 8-2　启动和停止页面

单击链接，进入该管理员功能，可以看到“Java Instances”“Java Services”和“Java Applications”3 个标签，分别列举了当前服务器中所有活动的服务器结点实例以及运行在对应实例上的所有 Java 服务和 Java 应用，如图 8-3 所示。

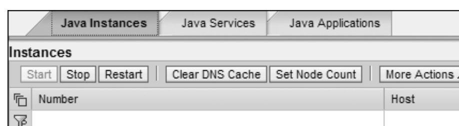


图 8-3　Java 实例管理界面

8.2.1　Java 服务器实例

Java 应用服务器支持服务器集群，每个服务器可以安装多个实例，不同的实例也可以安装在不同的物理服务器上，用于均衡负载。在 Java 实例页签中可以看到当前服务器所安装的所有实例，它们是应用集群的组成单元，能够单独地启动或停止，可以通过系统标识实例编号来进行区分，如需直接访问不同实例，除了实际 IP 有区别外，端口号也会根据安装时的实例编号有所不同。当然，也有很多时候，是在用户压力测试或上线使用后发现负载超过了现有服务器的硬件能力，随后对服务器进行横向地扩充以缓解服务器的运行压力。

当然其实在每个 SAP NetWeaver 服务器实例中又包含一个或多个服务器结点（Server Node），结点上的线程担负着所有 Java 应用的运行功能，实际的应用功能、功能请求处理都是在每一个结点线程上完成的，因此服务器线程是服务端的最小应用处理单元。

服务器集群的负载均衡一般由 Web 分发器（Web Dispatcher）完成，它负责将用户的所有请求在不同的服务器实例之间进行转发和均衡；在每个实例的内部，Java 分发器（Java Diapatcher）则承担起了将请求在不同服务器结点继续进行转发和均衡的职责，如图 8-4 所示。

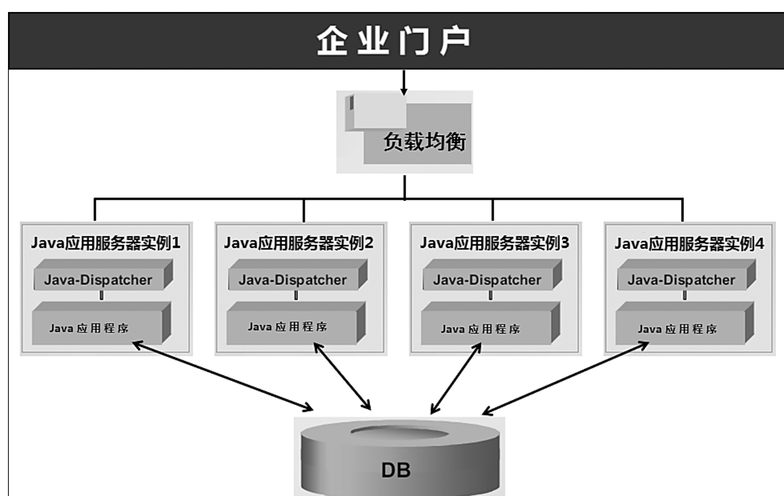


图 8-4　多实例情况的服务器架构

对于多个实例的 Java 集群（Cluster），SAP Web 分发器（Dispatcher）能够根据不同实例的实际使用情况，有效地分发请求到不同的 Java Dispatcher 下，使运行中的多个实例达到请求处理最大化。Web 分发器可以拒绝或接收来自客户端的请求，一旦接收就会平衡各个实例负载，因此 Web 分发器会在多个服务器实例的集群架构中使用，没有必要在单实例中运行 Web 分发器。

再往上一层，服务器集群（Cluster）是用于处理较大请求的服务器配置机制，当用户数据增多或者用户访问请求增大，应用系统不能满足负载时，可以通过对集群配置的扩容来加大应用系统对物理内存及 CPU 的可利用空间。服务器实例下的每一个结点对应 JVM 实例在运行，通过修改 JVM 的堆内存容量可以扩大 JVM 对物理内存的占用，从而进一步地增加可用的活动 JVM 进程，而集群又可以通过配置增加服务器实例来扩展集群中应用服务实例的数量，将硬件使用率最大化。在实际项目中往往因为实际的在线用户数大于预先设计的用户数量而导致服务器超荷，从而引发服务器宕机、系统崩溃等难预料的生产事故，在保证了硬件服务器的资源下，需要根据用户实际访问负荷对服务做出最合理的配置，以达到最优的请求处理性能。

回到 NWA 的页签中，在选中任意实例后可以通过单击“Start”“Stop”或“Restart”分别进行实例的启动、停止或重启，如图 8-5 所示。

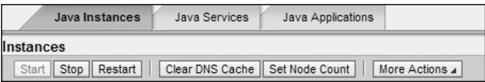


图 8-5　Java 实例的操作工具栏

8.2.2　Java 服务和应用程序

第二个页签是 Java 服务，可以看到当前服务器中所有的 Java 服务、服务组件名称以及当前运行状态。与 Java 实例页签中的功能类似，在 Java 服务页签中也可以通过按钮进行启动、停止以及重启等动作，如图 8-6 所示。

Java Instances　Java Services　Java Applications

Services

Start　Stop　Restart　More Actions

Name	Service Component Name	Status
ABAP Communicator	engine.sap.com/r3startup	Started
Accounting Infrastructure Service	sap.com/tc~bl~accounting_infra~srv	Started
Administration Adapter	sap.com/adminadapter	Started
Application Client	sap.com/appclient	Started
Application Configuration API	sap.com/tc~je~appconfiguration~api	Started
Application Locking	sap.com/applocking	Started

Accounting Infrastructure Service - Green LED; OK

图 8-6　Java 服务的操作界面

第三个页签是 Java 应用程序，可以看到当前服务器中所有的 Java 应用程序，其中“Vendor”为“sap. com”，即 SAP 提供的标准应用程序，而通过上方的筛选输入框也可以模糊查找用户自定义的应用。需要注意的是，只有状态为“Started”（默认使用绿灯图标标记）的应用才能够正常使用，所以当自定义开发的 Web Dynpro 应用程序、BPM 流程或 EJB 应用无法使用时，可以通过这里的状态监控查看是否正常运行，也可以通过按钮对应用进行重启，如图 8-7 所示。

在选中任一应用程序后，在下面还有与应用相关的一些明细信息可以进行配置，如依赖管理、模块、明细等内容，如图 8-8 所示。

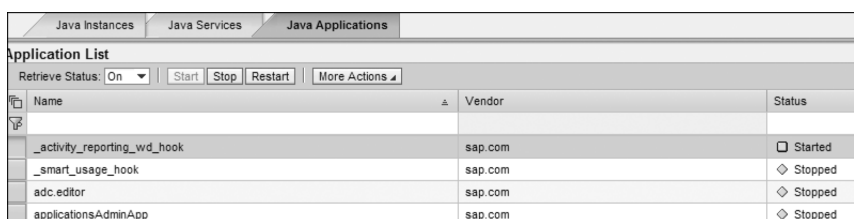


图 8-7　Java 应用程序的操作界面

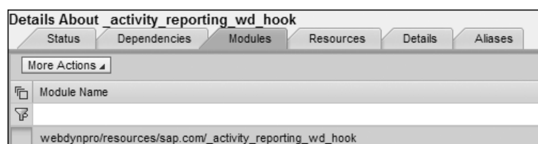


图 8-8　模块的明细信息

8.3　作业与定时任务

作业和定时任务是 SAP NetWeaver 平台的一个核心服务，运行在 Java 应用服务器上，在 ERP 系统（确切地说，是 ABAP 架构系统）中与之相对应的则是 CCMS 调度程序，可以通过“SM36”或“SM37”事务代码进行访问和使用。除了运行标准配置的定时任务以外，Java 应用服务器的定时作业服务还提供了第三方的 Java 定时作业与 SAP Java 应用服务器集成的功能。

SAP NetWeaver 平台为 Java 调度程序提供了低等级任务的调度配置功能，可以用于配置不同类型的定时任务，即在特定时间或以一定的周期在 Netweaver 服务器端后台自动地运行某些 Java 应用程序。从 Java 逻辑角度来理解，作业相当于一个 Java 类，而定时任务则是将作业实例化之后再添加运行规则产生的“实例”。

除了服务器端的支持，在开发工具 NWDS 中也添加了对定时作业下的支持，在创建对应的开发对象后，可以直接通过菜单自动在 EJB 工程中生成可部署的作业定义，省去了手动的配置环节。在将作业部署到服务端后，还可以通过 NWA 管理者平台配置定时作业的启动条件，另外 NetWeaver 也提供了定时作业相关的 API，可以在应用程序中直接编写动态启动条件的定时作业。

8.3.1　创建作业项目

下面介绍如何在 NWDS 中创建一个调度器作业项目。

1）首先创建一个新的调度器作业开发对象，进入 NWDS 开发者工作室。单击“File”→“New”→“Other”，在弹出的对话框中找到“Scheduler Job”选中，并单击“OK”按钮，如图 8-9 所示。

2）选择需要生成作业的 EJB 工程，填写包名、类名，任务名称和描述，随后单击“Finish”按钮完成任务创建，如图 8-10 所示。

3）打开 Java 编辑器，在自动生成的“StardardJob”类中编写作业运行时的逻辑，如图 8-11 所示。

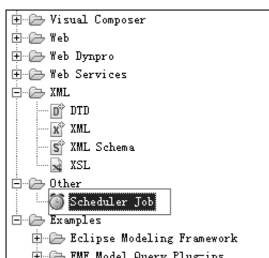


图 8-9　选择文件类型

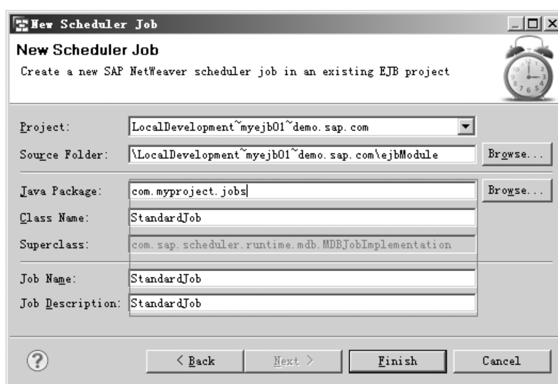


图 8-10　输入调度器作业信息

```
/**
 * Message-Driven Bean implementation class for: StandardJob
 *
 */
@MessageDriven(activationConfig = {
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "messageSelector", propertyValue = "JobDefinition=\'StandardJob\' " +
                "AND ApplicationName=\'demo.sap.com/myear01\'") })
public class StandardJob extends MDBJobImplementation {
    private static final com.sap.tc.logging.Location logger = com.sap.tc.logging.Location
            .getLocation(StandardJob.class);

    /**
     * Default constructor.
     */
    public StandardJob() {
        // TODO Auto-generated constructor stub
    }

    @Override
    public void onJob(JobContext jobContext) throws Exception {
        logger.infoT("Stardard Job Execution!");
    }

}
```

图 8-11　添加代码

4）接着修改对应的 xml 配置文件，在“ejb - j2ee - engine. xml”新增了对“StandardJob”类的描述属性。

```
<? xml version = "1.0" encoding = "UTF - 8"? >
<ejb - j2ee - engine xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance"
xsi:noNamespaceSchemaLocation = "ejb - j2ee - engine_3_0.xsd" >
<enterprise - beans >
<enterprise - bean >
<ejb - name > StandardJob </ejb - name >
<bean - props >
<property >
<property - name > destination - name </property - name >
<property - value > JobQueue </property - value >
</property >
<property >
<property - name > connection - factory - name </property - name >
<property - value > JobQueueFactory </property - value >
</property >
</bean - props >
</enterprise - bean >
</enterprise - beans >
</ejb - j2ee - engine >
```

5）接着创建一个“job - definition. xml”配置文件，并且 ear 描述文件中添加了以下描述。

```
<application - j2ee - enginexmlns:xsi = "http://www. w3. org/2001/XMLSchema - instance"
xsi:noNamespaceSchemaLocation = "application - j2ee - engine. xsd" >
<referencereference - type = "hard" >
<reference - targetprovider - name = "sap. com"  target - type = "service" > scheduler ~ runtime </
reference - target >
</reference >
</application - j2ee - engine >
```

6）完成配置代码后，编译并部署 EAR 工程至 Java 应用服务器，右击项目并选择“Development Component”→“Build”→“Deploy”，进行编译和部署，如图 8-12 所示。

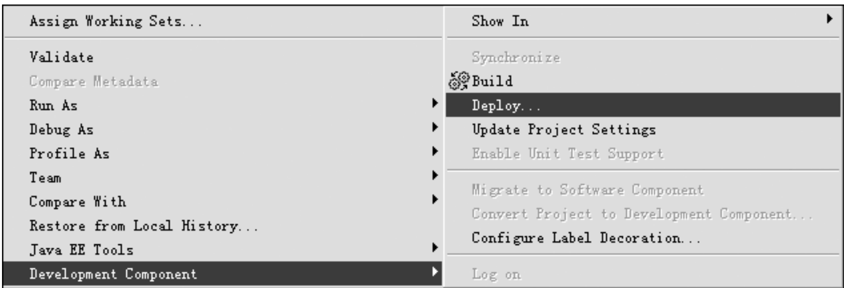


图 8-12　部署项目的菜单选择

7）接着使用管理员登录 NWA 平台，选择“操作”→“作业”→“Java 调度器”，如图 8-13 所示。

8）进入页面后，就能看到已经成功发布的对应的定时作业了，如图 8-14 所示。



图 8-13　NWA 平台的 Java 调度器

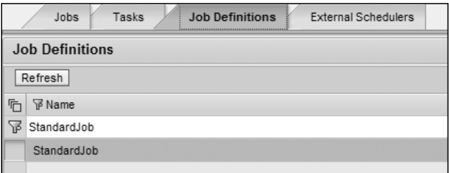


图 8-14　任务定义界面

8. 3. 2　NWA 配置定时任务

当在 NWA 中对定时任务进行启动和运行方式配置时，默认有 3 种时间规则：重复（recuring）、周期（Cron）和简单（Simple）。

- 重复：在配置的某一时间段，按照配置的发起间隔执行定时任务。例如，将开始日期（Start Date）设置为“2016 - 11 - 11”、结束日期（End Date）设置为“2017 - 11 - 11”、时期（Period）设为“3 s”，那么在“2016 - 11 - 11”到“2017 - 11 - 11”的这个时间段内每 3 秒执行一次定时任务。
- 周期：用于指定间隔的某一具体时间点执行定时任务。例如，年（Year）设置为“ * ”，表示每年运行，月（Month）设置为“8”，日（Day Of Month）设置为“8”，

时（Hour）设置为“8”，分（Minute）设置为“8”，那么在每年的 8 月 8 日 8 时 8 分执行一次对应的定时任务。

- 简单：设置一个未来时间点，并在这个时间执行，配置任务只会执行一次。例如，开始日期（StartDate）设置为“20161112”，开始时间（Start Time）设置为“18:08:09”，则服务器会在这个时间点执行定时任务。

下面在服务器端对发布的定时任务进行具体触发条件的配置过程。

1）使用管理员登录 NWA 平台，选择“操作”→“作业”→“Java 调度器”，如图 8-15 所示。

2）选中“Tasks”任务页签，选中需要进行配置的作业，随后单击上方的“添加”按钮，如图 8-16 所示。



图 8-15　NWA 平台的 Java 调度器

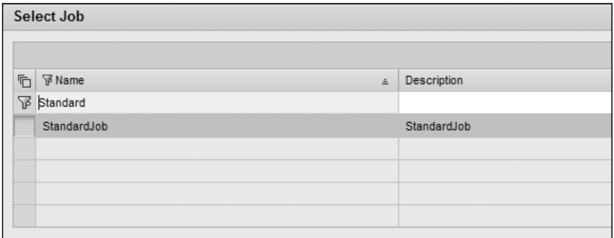


图 8-16　选择任务

3）设置定时任务的基本属性，如图 8-17 所示。

4）设置其他参数和运行规则，这里以简单规则为例进行设置，如图 8-18 所示。

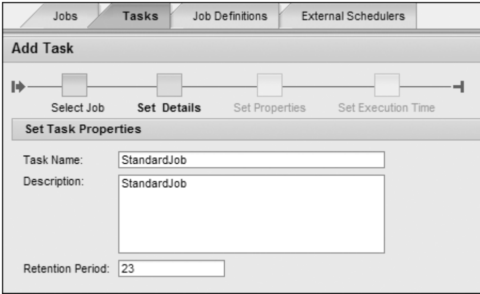


图 8-17　设置任务属性

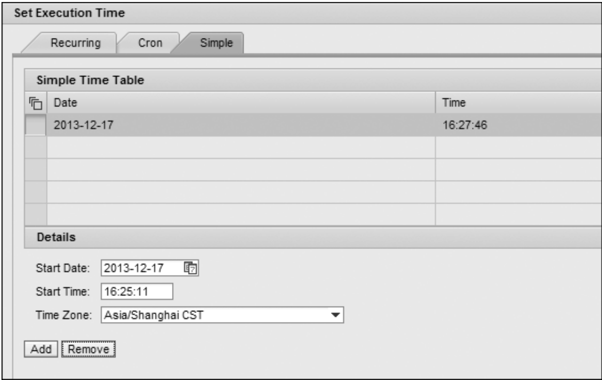


图 8-18　设置执行时间

5）按照规定的时间规则，查看生成的 Jobs，如图 8-19 所示。

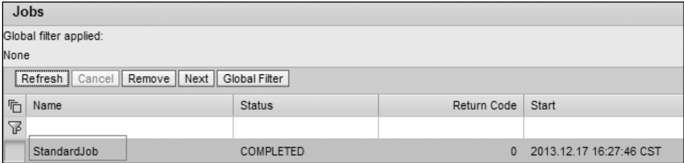


图 8-19　查看执行状态

6）同时在日志查看器中也生成了对应的运行日志，表示该任务运行成功，如图 8-20 所示。

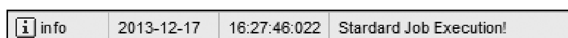


图 8-20　运行日志

8.3.3　WD4J 创建定时任务

除了上面的通过 NWA 后台配置方式设置定时任务的运行规则外，SAP 还提供了调度器的 API，允许开发人员通过代码端直接访问 API 并定义时间规则启动相应的定时任务。

开发实例：

1）首先打开一个 Web Dynpro 项目，并在视图中创建一个触发按钮，如图 8-21 所示。

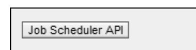


图 8-21　运行效果

2）定义按钮的默认动作，并在 Java 编辑器中添加动作的处理逻辑：

```
//通过 JNDI 获取 scheduler 引用
Scheduler scheduler;
try {
   InitialContext ctx = new InitialContext( ) ;
   scheduler = ( Scheduler) ctx. lookup( "scheduler" ) ;
} catch ( NamingException e) {
wdComponentAPI. getMessageManager( )
. reportException( e. getMessage( ) ) ;
   return ;
}
//获取 JobDefinition
JobDefinition helloJobDef = scheduler
. getJobDefinitionByName( "StandardJob" ) ;
//创建时间规则,Recurring Time,开始时间与结束时间相同
Calendar calendar = Calendar. getInstance( ) ;
//时间点为一分钟
calendar. add( Calendar. MINUTE,1) ;
java. util. Date date = calendar. getTime( ) ;
//通过 calendar 创建schedulertime
SchedulerTime time = new SchedulerTime( date,TimeZone. getDefault( ) ) ;
//定义时间规则
RecurringEntry[ ] re = { new RecurringEntry( time) } ;
//创建 Task,输入参数包含了 Job 定义、Job 参数、通过代码构造的时间规则,以及 Task 描述
SchedulerTask task = SchedulerTask. createSchedulerTask( helloJobDef
. getJobDefinitionId( ) ,null ,re,null ,null , -2,
"StandardJob Task" ,"StandardJob Task Description" ,null ) ;
//执行 Task
try {
   scheduler. schedule( task) ;
} catch ( TaskValidationException e) {
   wdComponentAPI. getMessageManager( )
   . reportException( e. getMessage( ) ) ;
   return ;
}
wdComponentAPI. getMessageManager( ). reportSuccess( "定时作业启动成功!" ) ;
//注:该 DC 需要添加对 tc/je/scheduler/api 的依赖。
```

3）编译、部署、运行，单击按钮，如图 8-22 所示。

定时作业启动成功!

图 8-22　提醒消息

4）使用管理员登录 NWA，单击“操作”→“作业”→“Java 调度器”，就可以查看通过代码产生的定时任务及其触发规则，如图 8-23 所示。

5）进入 Java 调度器也可以看到具体任务的执行情况，如图 8-24 所示。

Tasks

Refresh　Add　Remove　Hold　Release

Name	Description
Stand	
StandardJob Task	StandardJob Task Description

图 8-23　任务操作界面

Jobs

Global filter applied:
None

Refresh　Cancel　Remove　Next　Global Filter

Name	Status	Return Code
Stand		
StandardJob	COMPLETED	0

图 8-24　查看任务执行状态

注：一般调度器的 API 在定时任务参数或触发条件不确定，即需要动态设置的情况下使用，同时 API 也可以构造复杂的时间规则，但不管以何种方式配置，都要及时进行管理和监控。

8.4　Adobe 技术交互表单

Adobe 技术的 SAP 交互表单是 SAP 和 Adobe 公司为了提供更好的表单创新以及可操控性联合提出的一个表单解决方案。这一解决方案与 SAP 的设计时以及运行时环境高度集成，可以保证 Adobe 标准的表单在 SAP 相关应用程序中的正常浏览、使用甚至是业务数据交互。它可以在 SAP 系统的下列场景中被定义和使用：

1）在表单编辑器中或是 ABAP 工作台提供表单制作，Adobe 技术的表单可以被用以打印、归档或是作为电子邮件直接进行发送。

2）在 Web Dynpro 的开发环境中（包括 Web Dynpro ABAP 和 Web Dynpro Java）作为在线的交互式表单进行开发和制作，可用于 Web Dynpro 应用程序的页面打印或数据交互。

3）通过在 NetWeaver 平台预先安装 ADS 服务的环境中，通过 LCD 编辑器进行文档模板的统一制作。

从 WD4J 的开发技术角度来说，ADS（Adobe Document Services）是运行在 NetWeaver Java 应用服务器之上的 Adobe 服务。WebDynpro 应用程序通过调用 ADS 在运行环境生成通过 LiveCycle Designer 编辑器制作的页面模板，表单设计时可以绑定 Web Dynpro 应用程序中的上下文，以实现实时数据的展现与交互，如图 8-25 所示。

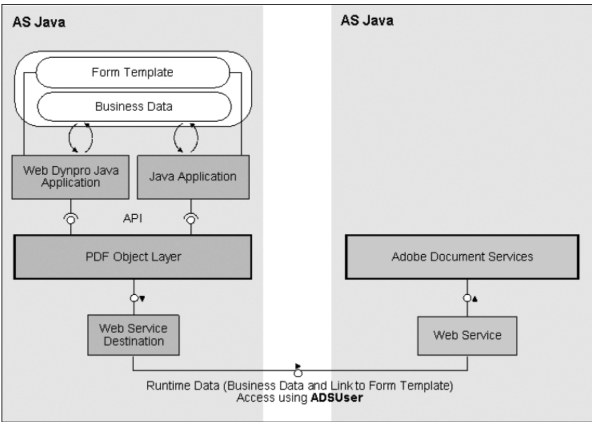


图 8-25　WD4J 与 ADS 交互的架构图

8.4.1　配置本地 LCD

对于访问应用程序以及操作使用表单的最终用户来说，客户端能够查看 Adobe 表单需要

提前安装 Abode Reader 软件，因此使用 WebDynpro 打印 Adobe 表单时，需要在本地安装对应版本的 Adobe Reader 产品。

对于开发人员而言，在本地开发客户端和个人计算机中，需要预先安装 Adobe LiveCycle Designer（LCD）。LCD 是 Adobe 公司提供的 PDF 编辑器，SAP 的开发与运行环境都实现了与该工具的集成，客户端安装 LCD 后，开发人员可以直接通过 NWDS 中编辑视图元素来创建和修改自定义的交互表单。

LCD 同时支持 NWDS 开发与 ABAP 工作台。SAP GUI 客户端已经集成了对 LCD 的集成，但需要注意的是，SAP GUI 7.30 以上的版本才支持 Designer 9.0。随着 NetWeaver 平台的不断升级，目前 LiveCycle Designer 9.0 支持 NetWeaver 7.30 及以上版本的 ADS。LiveCycle Designer、Adobe Reader 以及 NetWeaver 三者的版本只有保持一致，才能够正确地使用。表 8-1 所示为 NetWeaver 的不同版本以及能够支持的 LiveCycle Designer 版本信息。

表 8-1　NetWeaver 不同版本的支持信息

LiveCycle Designer	NetWeaver	Adobe Reader
8.1	7.0 SP13 及以上	7.05 及以上
8.2	7.20 及以上	8.1 及以上
9.0 系列	7.30 及以上	9.0 及以上
9.8	7.30 及以上	9.1 及以上
10.0	7.30 及以上	9.1 及以上
8.6		9.1 及以上

通过 SAP 的官方支持网站就可以下载对应版本的 LiveCycle Designer（访问地址参考：http://service.sap.com/installations）

打开链接后目录参考："Installations and upgrades" → "Browse our Download Catalog" → "SAP NetWeaver and complementary products"，如图 8-26 所示。

objects are available for download:

File Type	Download Object	Title	Info File	File Size [kb]
ZIP	50120297_1	NW 7.0 Presentation - Adobe LiveCycle Designer 9.0	Info	459151

图 8-26　下载安装文件

在 WD4J 的开发过程中，只有安装 Adobe LCD 表单设计器的插件，才能通过 NWDS 直接编辑 Adobe 交互表单。LCD 设计器也提供了多种不同表单制作控件，这一点与 WebDynpro 视图元素的设计思路相同，既能够满足复杂多样的表单格式设计，又可以直接通过设计器将表单控件与 Web Dynpro 上下文进行绑定，从而实现动态实时表单的功能。

在正确安装后，打开 NWDS 中任意的 Web Dynpro 项目，即可添加交互表单对应的视图元素"Interactive Form"，如图 8-27 所示。



图 8-27　视图的预览效果

在选择编辑该元素后，就能通过 LCD 设计器对表单进行详细的设计和开发了。图 8-28 是默认的设计器使用界面，上方是设计器的工具栏，左侧是元素清单的结构图（类似于 Web Dynpro 中 Outline 的元素清单）；中间是设计视图，显示了表单的实际展现效果；右侧是图形化的控件使用以及相关控件属性的详细配置视图，如图 8-28 所示。

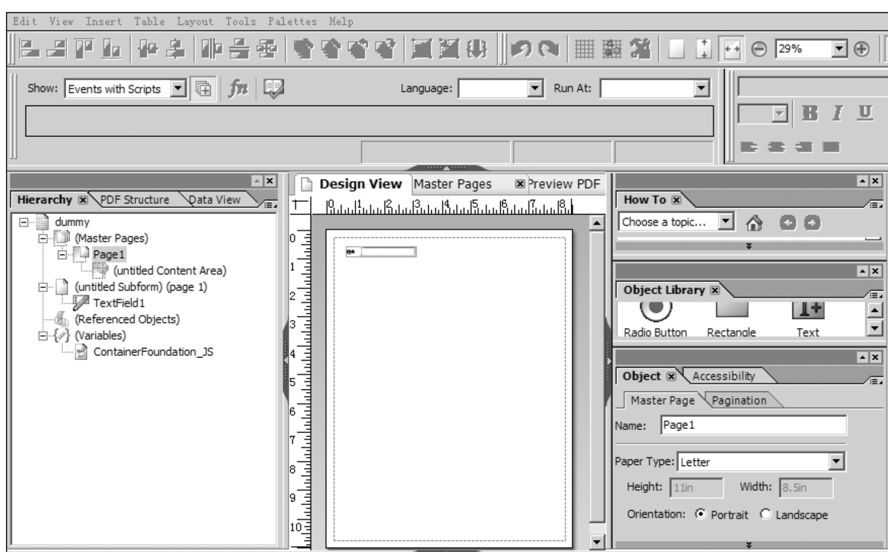


图 8-28　设计器界面

8.4.2　配置 ADS 服务

在完成了开发人员和用户端的本地配置后，需要对 SAP NetWeaver 中的 ADS 服务进行配置，以便正确提供对于 Adobe 表单运行时的支持。基础配置包括创建验证用户、建立 PDF 对象层到 ADS 的权限认证。NWA 提供了在线自动执行的配置功能，如果后台执行失败，则自定义参数时可以使用手工配置。以下是详细的配置过程。

配置实例：

1）检查当前服务器环境是否安装有 ADS 服务，访问地址为：http://<host>:<port>/sld/fun。如尚未安装对应服务的环境，则需先进行安装文件的下载和安装工作，如图 8-29 所示。

Functional Unit	Usage Type
Adobe Document Services (not installed)	Adobe Document Services
Advanced Adapter Engine (AF) (not installed)	PI Adapter Engine
Advanced Adapter Engine Extended (AEX) (not installed)	Advanced Adapter Engine Extended or Process
BI Java (not installed)	BI Java
CM Services (Change Management Services) (not installed)	Application Server Java

图 8-29　检查已安装服务

2）创建验证用户。使用管理员账号创建一个验证用户，在“登录标识”文本框中输入“ADSUser”，在“安全策略”下拉列表中选择“技术用户”（这是针对后台服务的用户类型，不需要修改登录密码，避免连接错误），如图 8-30 所示。

3）分配 ADS 所需要的角色给该用户，可直接搜索“SAP_ADSCALLER”，不需要额外创建，如图 8-31 所示。

登录标识：* ADSuser
定义初始密码
生成密码
禁用密码
定义密码：* ••••••
确认密码：* ••••••
姓氏：* ADS
名字：
电子邮件地址：
称谓：
语言：
激活可存取性功能：
安全策略： 技术用户
唯一标识：

图 8-30　创建新用户



图 8-31　搜索角色

4）创建 ADS 服务目标，使用管理员账号登录 http://<host>:<port>/nwa 管理者平台，选择“SOA”工作中心，并找到“技术配置”（Technical Configuration）中的“目标模板管理”（Destination Template Management），打开链接后，单击“新建”按钮，新建目标，如图 8-32 所示。

Enter a name for the logical port
Destination Type: WSIL
Destination Name: * ConfigPort_Document
URL: * http://<host>:<port>/inspection.wsil
Socket Timeout (in Milliseconds): 60,000
System: Java ABAP
System Name:
Host Name:

图 8-32　输入目标信息

5）完成基本信息填写后，单击“下一步”按钮，进行认证配置，输入之前创建的用户“ADSUser”及其密码，随后单击“完成”按钮，如图 8-33 所示。

Authentication
Authentication: HTTP Authentication
User ID/Password (Basic)
User ID/Password (Digest)
X.509 Client Certificate
Logon Ticket
SAML Assertion
Details Off
Details
User ID/Password
User ID: ADSUser
Password: ••••••
Confirm Password: ••••••

图 8-33　输入认证信息

6）完成创建后，就可以在目标表格中看到对应的项目并使用了，如图 8-34 所示。

New Delete
Destination
Config
ConfigPort_Document

图 8-34　目标清单

7）除了手动填写配置外，NWA 还提供了在线配置工具，可以使上述过程在后台自动完成，过程执行完后系统默认创建对应用户“ADSUser”以及目标“ConfigPort_Document”，

如图 8-35所示。

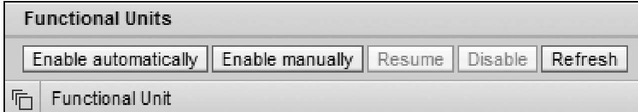


图 8-35　自动配置的操作界面

8.4.3　WD4J 开发交互表单

在完成了本地配置以及服务器端配置后，就可以通过 NWDS 进行表单的开发以及使用了，具体过程如下。

1）在 Web Dynpro 项目中打开任意一个视图，首先创建表单所需要的上下文结点和特性。创建一个集合属性为“1··1”的“PrintNode”结点，并包含一个“header”结点和一个“contentList”，集合属性分别为“1··1”和“0··n”，用于存放对应的表头信息和行项目信息。随后创建对应字段的相关特性，如图 8-36 所示。

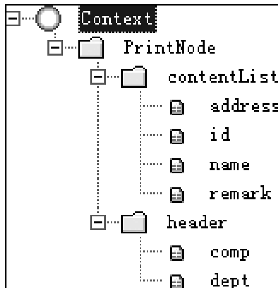


图 8-36　上下文清单

2）右击“Outline”中的“RootElement”根结点元素，选择添加子元素，在弹出的对话框中选择“InteractiveForm”类型元素，并单击“OK”按钮，如图 8-37 所示。

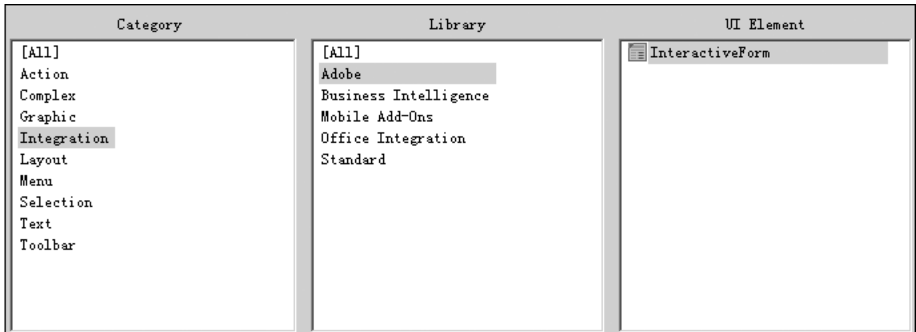


图 8-37　选择元素类型

3）添加完成后即可看到对应的元素对象，如图 8-38 所示。

4）选中“InteractiveForm”，将属性中的“dataSource”数据源指定上下文中的“PrintNode”结点，保存所有修改，如图 8-39 所示。

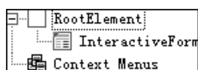


图 8-38　元素清单

contextMenuId	
dataSource	PrintNode
displayType [Deprecated]	activeX*
enabled	true

图 8-39　修改属性

5）右击“Outline”中的“InteractiveForm”，在弹出快捷菜单中选择“Edit”，对表单进行编辑，如图 8-40 所示。

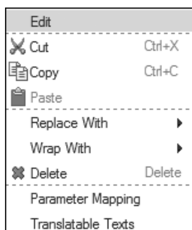


图 8-40　编辑对象的菜单选择

6）进入 Adobe LCD 编辑器，如图 8-41 所示。

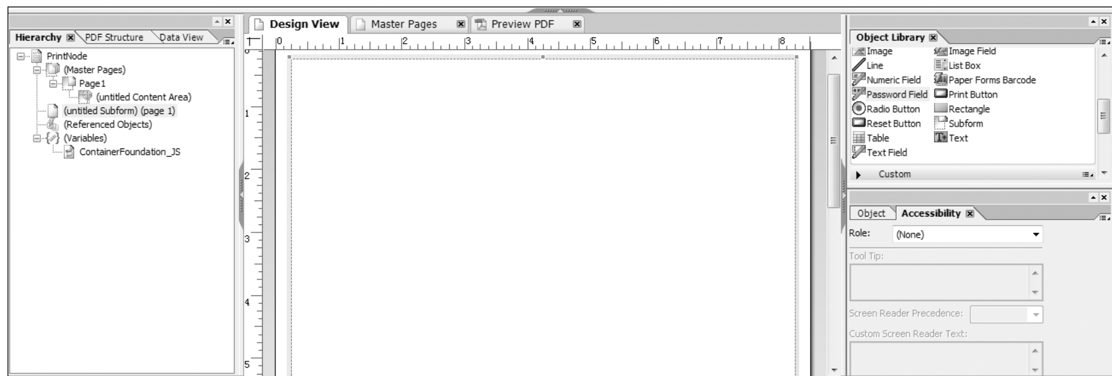


图 8-41　LCD 编辑器界面

7）最左侧的窗口中默认显示“Hierarchy”视图，用于展现设计视图的页面元素层次结构，包括主页面、页面、子表单等内容。在子表单（Subform）中编辑页面需要的控件，如图 8-42 所示。

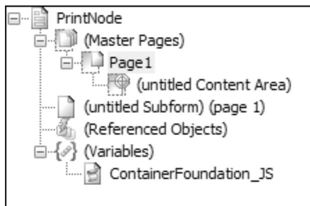


图8-42　编辑器中的元素清单

8）最左侧窗口中隐藏的“DataView”用于查看绑定的数据源对应的结点结构，如图 8-43 所示。

9）最右侧的窗口默认显示“Object Library”控件库，包含了所有可编辑的表单控件，根据不同类型放在不同的子标签中，类似于 Web Dynpro 视图中的 UI 元素库，如图 8-44 所示。

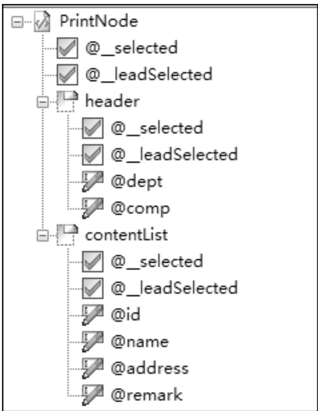


图 8-43　结点结构

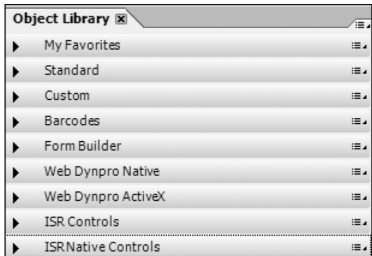


图 8-44　UI 元素库

10）其中“Standard”标准对象中包含了一些常用的控件，如图 8-45 所示。

11）右下方窗口则显示了视图控件的详细属性，通过 Object 修改控件各种属性，使页面的风格统一。根据控件类型的不同，属性分为不同的页签，其中“Binding”用于标识绑定该控件对应的数据源，即 Web Dynpro 上下文对象，如图 8-46 所示。

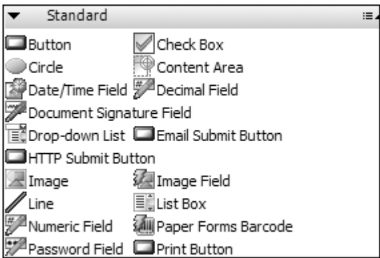


图 8-45　常用控件

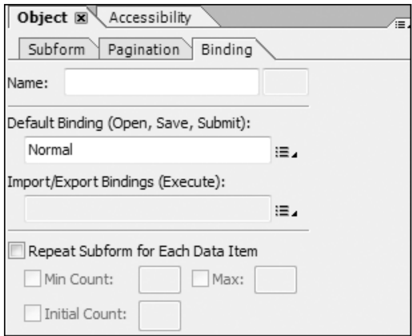


图 8-46　控件的详细属性

12）编辑器的中间窗口则用于视图的设计和开发，隐藏的“Preview PDF”视图用于设计模板的离线预览，可以随时查看开发页面的实际显示效果，如图 8-47 所示。

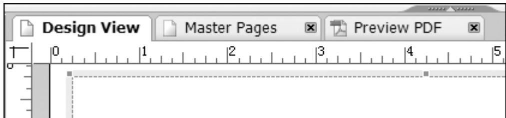


图 8-47　设计视图与预览视图

13）下面开始对表单页面内容的编辑开发。其中，表头显示公司与部门，使用“TextField”控件，内容则以表格方式显示人员列表属性。首先从“Object Library”中选择“TextField”并拖曳至“Disign View”设计视图中，如图 8-48 所示。

14）选择该控件对其属性进行调整，将文本修改为“公司”，如图 8-49 所示。

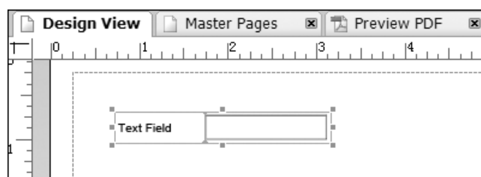


图 8-48　添加后效果

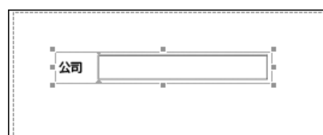


图 8-49　修改文本属性

15）随后在右下方找到“Binding”中的“Default Binding”，单击右侧按钮并找到对应的上下文特性，对控件和上下文进行绑定，如图 8-50 所示。

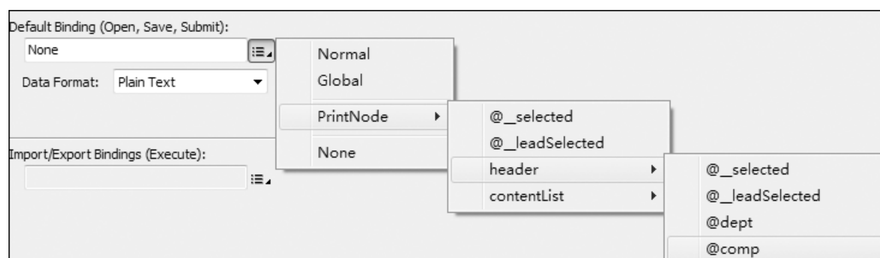


图 8-50　绑定上下文的菜单选择

16）在弹出的对话框中选择是否更新相关属性，选择不更新相关属性，如图 8-51 所示。

17）完成了对应控件与 Web Dynpro 上下文的绑定，如图 8-52 所示。



图 8-51　绑定属性对话框

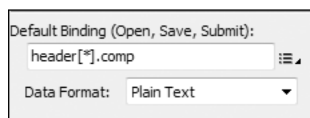


图 8-52　完成绑定后的界面

18）按照相同的方法添加部门输入框控件，并绑定对应的上下文特性，如图 8-53 所示。

19）随后设置字体格式，选择标签字体即可进行设置。将“Hierarchy”视图中的对象批量选中，随后修改字体为“Adobe Song Std. L”，如图 8-54 所示。



图 8-53　创建后的效果

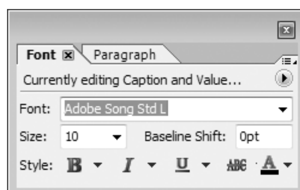


图 8-54　修改字体

20）可以单击“Preview PDF”进行PDF文件预览，查看预览效果，如图8-55所示。



图8-55　预览效果

21）修改控件显示样式，在“Hierarchy”视图中批量选中新建的控件，在Object视图中找到“Field”页签中的“Appearance”属性，选择“None”，如图8-56所示。

22）单击“Value”页签，在“Type”下拉列表中选择“Read Only”，如图8-57所示。

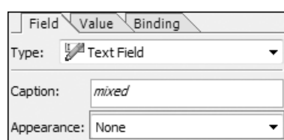


图8-56　修改属性1

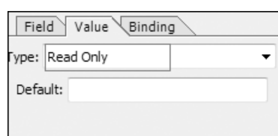


图8-57　修改属性2

23）再次进行PDF预览，确认实际效果与预期相符，如图8-58所示。



图8-58　预览效果

24）接着在设计视图中插入表格，设置列数与默认行，单击“OK”按钮，如图8-59所示。

25）在设计视图中的效果如图8-60所示。

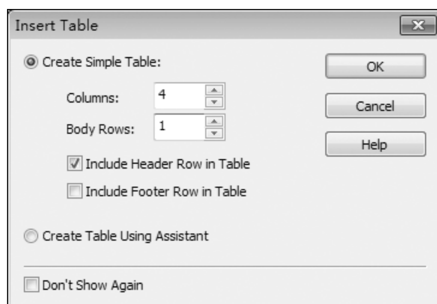


图8-59　插入表格对话框

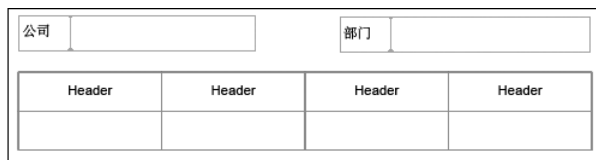


图8-60　预览效果

26）编辑表头，设置表头的描述以及对应字体，如图 8-61 所示。

人员编号	姓名	地址	备注

图 8-61　编辑文本属性

27）选中表格，在右下属性中找到“Binding”→“Default Binding”，进行上下文绑定，如图 8-62 所示。

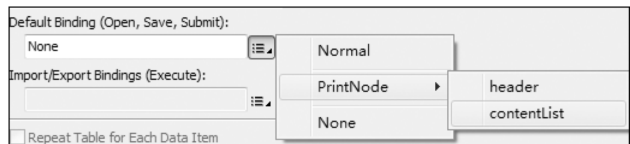


图 8-62　绑定上下文的菜单选择

28）批量选择单元格对象，选择“Text”类型，如图 8-63 所示。

29）选中每一个单元格，并分别绑定 Web Dynpro 的上下文，如图 8-64 所示。

30）接着选中表格中的“Row”行，在右下“Object”对象属性的“Binding”中选中“Repeat Row for Each Data Item”复选框，用于显示多条的行项目数据，如图 8-65 所示。

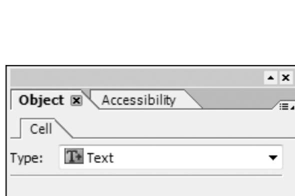


图 8-63　修改属性 1

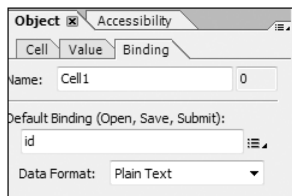


图 8-64　修改属性 2

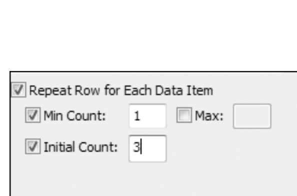


图 8-65　修改属性 3

31）PDF 预览查看确认表单的实际效果，如图 8-66 所示。

公司		部门	
人员编号	姓名	地址	备注

图 8-66　预览效果

32）在完成表单页面的开发后，回到 Web Dynpro 的项目中，进入视图的 Java 编辑器，在初始化方法中添加测试用例的代码，即可编译运行：

```
publicvoid wdDoInit( )
    {
//@ @ begin wdDoInit( )
//赋值表头
    IHeaderElement currentHeaderElement = wdContext. nodeHeader( )
```

```
                .currentHeaderElement();
                currentHeaderElement.setComp("SAPEYES 公司");
                currentHeaderElement.setComp("SAPEYES 部门");
        //赋值表内容
                IContentListNode nodeContentList = wdContext.nodeContentList();
                IContentListElement createAndAddContentListElement =
nodeContentList.createAndAddContentListElement();
                createAndAddContentListElement.setId("001");
                createAndAddContentListElement.setName("Wang");
                createAndAddContentListElement.setAddress("地址 01");
                createAndAddContentListElement.setRemark("备注 01");
                createAndAddContentListElement =
nodeContentList.createAndAddContentListElement();
                createAndAddContentListElement.setId("002");
                createAndAddContentListElement.setName("Zhang");
                createAndAddContentListElement.setAddress("地址 02");
                createAndAddContentListElement.setRemark("备注 02");
                createAndAddContentListElement =
nodeContentList.createAndAddContentListElement();
                createAndAddContentListElement.setId("003");
                createAndAddContentListElement.setName("Li");
                createAndAddContentListElement.setAddress("地址 03");
                createAndAddContentListElement.setRemark("备注 03");
//@@end
        }
```

8.5 自定义数据源

在 Web Dynpro 开发中，大部分的数据源都通过模型导入接口的方式将原始数据传递给 Web Dynpro 的应用程序。但是在很多企业中，保留了很多原有的系统，可能对于开发接口存在人员或技术上的不足。对于这样的业务场景，SAP 在 NetWeaver 平台中提供了配置 JDBC 驱动的方式进行数据库的直接访问，这也是 Web Dynpro 应用程序中直接操作数据库的常用方式，数据源的配置可以脱离与代码开发的紧耦合关系：一方面，数据的连接信息不需要通过硬编码写入应用程序，在数据库迁移或应用程序迁移涉及连接信息修改时，不需要再次修改应用程序的开发代码，保证了程序代码的完整可靠；另一方面，数据库的连接数可以进行实时在线调配，根据实际系统的负荷以及用户访问量随时随地通过 NWA 平台进行修改。从项目实施客户角度来看，数据库连接信息也可以保持与代码的独立性，不再需要担心客户数据库相关信息面向开发人员的公开和小范围分享，从某种程度来说，提升了数据库的保密性和安全性，降低了对原有系统的影响和风险。

下面通过一些开发和配置实例进行实践，具体包括创建数据库连接驱动、配置自定义数据源以及访问自定义数据源几个方面。

8.5.1 创建连接驱动

1）首先使用管理员账号登录 SAP NetWeaver 系统的 NWA 平台，接着进入“Configuration”→“Infrastructure”→“Application Resources”，在“Show”下拉列表中选择“JDBC Drivers”，如图 8-67 所示。



图 8-67　查看 JDBC 驱动

2）单击“Create New Resource”按钮，在弹出的菜单中选择“DeployNew JDBC Driver”。驱动的 JAR 包是对应类型数据库连接的根本，只有先上传指定版本的类库，才能创建对应的数据库连接，如图 8-68 所示。

3）接着输入自定义驱动名称“Ora_Driver”，用于连接甲骨文数据库，单击“Add New Driver File”按钮，如图 8-69 所示。

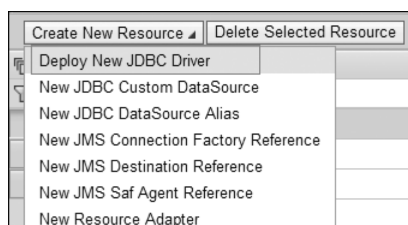


图 8-68　创建资源的类型选择

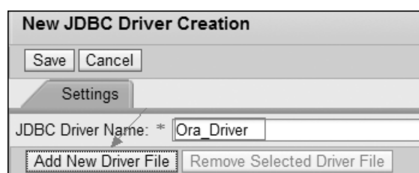


图 8-69　添加新的驱动文件

4）在弹出的对话框中选择本地的数据库驱动文件，单击“OK”按钮进行上传，这里以甲骨文数据库的驱动文件为例，如图 8-70 所示。

5）单击“Save”按钮，完成配置后，在清单中就能看到新建的连接驱动了，前面绿色的图标表示状态正常运行，如图 8-71 所示。

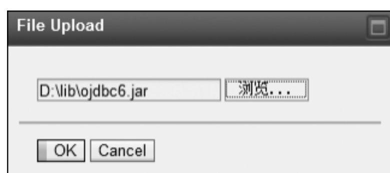


图 8-70　选择对应的驱动文件

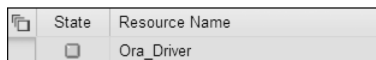


图 8-71　查看资源状态

6）单击“Create New Resource”，在弹出的菜单中选择“New JDBC Custom DataSource”，创建新的 JDBC 自定义数据源，如图 8-72 所示。

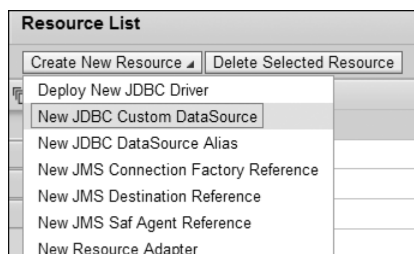


图 8-72　创建新资源的类型选择

7）在随后的页面中填入具体的自定义数据源相关配置信息，如图 8-73 所示。配置信息示例见表 8-2。

表 8-2　配置信息示例

配 置 属 性	配 置 说 明	配 置 示 例
Application Name	应用名称	PRO_DS
DataSource Name	数据源名称	PRO_DS
Driver Name	驱动名称	Ora_Driver（选择之前上传驱动）
SQL Engine	SQL 引擎	Vendor SQL
Isolation Level	独立等级	Default
JDBC Version	JDBC 版本	1x（without XA support）
Driver Class Name	驱动类名称	oracle. jdbc. OracleDriver
Database URL	数据库 URL 链接	jdbc：oracle：thin：@host：port：servicename
User Name	连接用户名	name
Password	连接用户密码	pass

图 8-73　输入资源信息

8）填写完毕后，单击“Save”按钮进行保存，在自定义数据源的清单中就能看到对应的“PRO_DS”数据源，如图 8-74 所示。

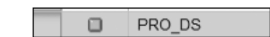


图 8-74　查看资源运行状态

8. 5. 2　配置自定义数据源

在完成数据库驱动上传以及数据源的创建以后，下面对该自定义数据源进行更详细的配置，以便后续的开发使用。

1）进入“JDBC Custom DataSources”自定义数据，选中创建的“PRO_DS”，进入明细显示页面，可以看到有“Settings”“Connection Pooling”和“JDBC Driver”等配置内容。在第一个“设置”选项卡中，可以对自定义数据源的一些基本信息进行修改，如图 8-75 所示。

2）在“Connection Pooling”选项卡中可以设置初始化连接数、最大连接数与获取连接超时时间。默认使用初始连接数“0”，最大连接数“25”，最长等待时间“120”，如图 8-76 所示。

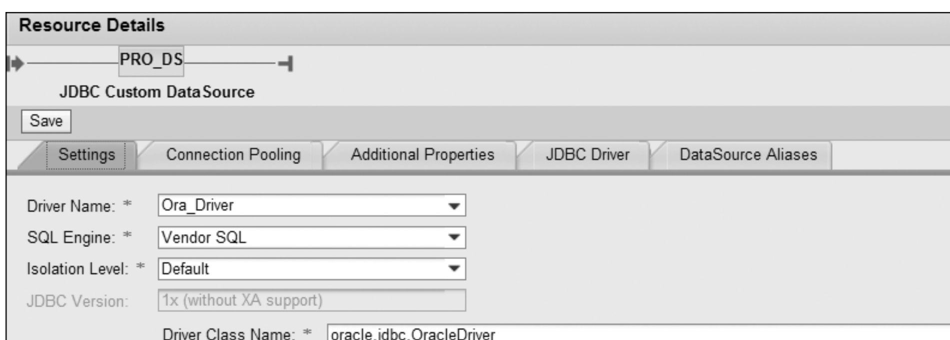


图 8-75　设置明细信息

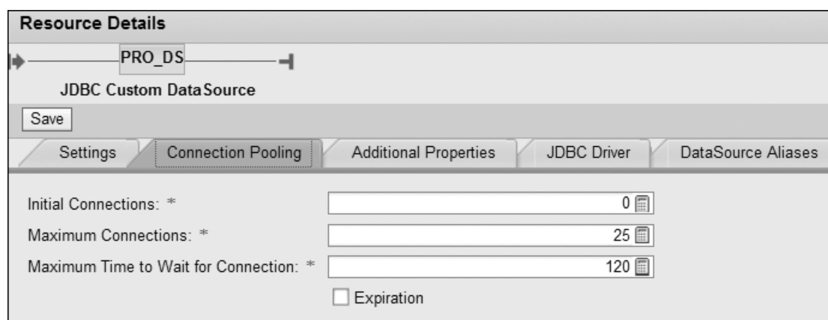


图 8-76　设置连接池信息

3）在“JDBC 驱动”选项卡中可以更新该自定义数据源使用的驱动，如果类型匹配错误，则将无法正常使用，如图 8-77 所示。



图 8-77　查看驱动资源

8.5.3　WD4J 连接实例

1）首先在 NWDS 中创建一个 Java 的 DC 项目，新建一个“Properties”，如图 8-78 所示。

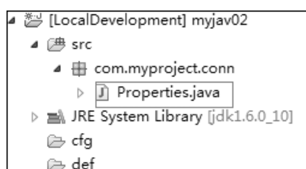


图 8-78　创建属性类

2）添加以下代码至属性类中：

```
public static String DATASOURCE = "jdbc/PRO_DS";//CUSTOM DS
public static String LOCATION_BASIS = "com. myproject";
public static String DATASOURCEXCEPTION = "获取数据源失败,请检查数据源是否已启动";
public static String DATABASEXCEPTION = "数据库连接异常,请联系系统管理员";
public static String UNKWONEXCEPTION = "未知错误,请联系系统管理员";
/ * *
 * return datasource
 * /
public static String getDataSource( ) {
        return DATASOURCE;
        }
/ * *
 * 根据消息类型返回错误消息
 * /
public static String getExceptionMsg( String key) {
    if( key. equalsIgnoreCase( "DATASOURCEXCEPTION" ) ) {
        return Properties. DATASOURCEXCEPTION;
    } else if( key. equalsIgnoreCase( "DATABASEXCEPTION" ) ) {
        return Properties. DATABASEXCEPTION;
    } else{
        return Properties. UNKWONEXCEPTION;
    }
}
```

3）新建“DBConnection”数据库连接类，如图 8-79 所示。



图 8-79　创建新的类

4）添加获取数据库连接的代码至连接类中。

```
private static final com. sap. tc. logging. Location logger =
com. sap. tc. logging. Location. getLocation( DBConnection. class);
private static final String DATABASE_JNDI_PROVIDER = "com. sap. engine. services. jndi. InitialCon-
textFactoryImpl";
private static Context getInitialContext( )  throws Exception{
try{
    java. util. Properties properties = null;
    properties = new java. util. Properties( );
    properties. put( Context. INITIAL_CONTEXT_FACTORY, DATABASE_JNDI_PROVIDER);
    return new InitialContext( properties);
} catch( Exception e) {
    logger. errorT( Properties. getExceptionMsg( "DATASOURCEXCEPTION" ) + e. getMessage( ) );
    throw new Exception( Properties. getExceptionMsg( "DATASOURCEXCEPTION" ) ,e);
}
}

/ * *
 * 根据 jndi 名,获得数据库连接
 * @ return
 * @ throws NamingException
```

```
 * @ throws SQLException
 * /
public static Connection getConnection( ) throws Exception{
      try{
            //name = Propertise. getDataSource( ) ;
            Connection conn = null;
            InitialContext jndiCntx = ( InitialContext) getInitialContext( ) ;
            DataSource ds = ( DataSource) jndiCntx. lookup( Properties. getDataSource( ) ) ;
            conn = ds. getConnection( ) ;
            return conn;
      } catch( Exception e) {
            logger.
            errorT( Properties. getExceptionMsg( "DATASOURCEEXCEPTION" ) + e. getMessage( ) ) ;
            throw new Exception( Properties. getExceptionMsg( "DATASOURCEEXCEPTION" ) ,e) ;
      }
}
/ * *
 * @ param conn
 * @ param stmt
 * @ param rs
 * /
public static void freeConnection( Connection conn, Statement stmt,
ResultSet rs) {
      try{
            freeConnection( rs) ;
            freeConnection( stmt) ;
            freeConnection( conn) ;
      } catch( Exception ex) {
            logger. errorT( "释放连接时出错!" + ex. getMessage( ) ) ;
      }
      }
      public static void freeConnection( Connection conn, Statement stmt) {
      try{
            freeConnection( stmt) ;
            freeConnection( conn) ;
      } catch( Exception ex) {
            logger. errorT( "释放连接时出错!" + ex. getMessage( ) ) ;
      }
}
public static void freeConnection( Statement stmt, ResultSet rs) {
      try{
            freeConnection( rs) ;
            freeConnection( stmt) ;
      } catch( Exception ex) {
            logger. errorT( "释放连接时出错!" + ex. getMessage( ) ) ;
      }
}
public static void freeConnection( Connection conn) {
      try{
            if( conn! = null)
            conn. close( ) ;
      } catch( SQLException ex) {
            logger. errorT( "释放连接时出错!" + ex. getMessage( ) ) ;
      }
}
```

```
public static void freeConnection(Statement stmt){
    try{
        if(stmt!=null)
        stmt.close();
    }catch(SQLException ex){
        logger.errorT("释放连接时出错!"+ex.getMessage());
    }
}
public static void freeConnection(ResultSet rs){
    try{
        if(rs!=null)
        rs.close();
    }catch(SQLException ex){
        logger.errorT("释放连接时出错!"+ex.getMessage());
    }
}
```

5）接着创建实体 Bean“EntityBean”和“EntityDao”访问数据库，分别建立“User”与“UserDao”类，如图 8-80 所示。

6）在“User”类中添加相应的字段属性和默认的“get”以及“set”构造方法，如图 8-81 所示。

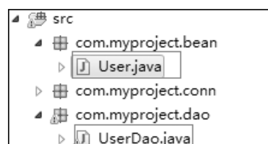


图 8-80　创建新的类

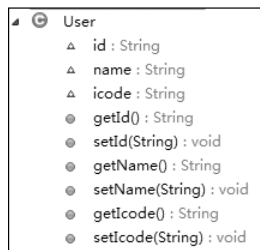


图 8-81　创建字段对应的构造方法

7）在“UserDao”中添加以下方法：

```
/**
 *查询用户
 *
 *@return
 */
public static List findUser(){
Connection conn=null;
PreparedStatement prepareStatement=null;
ResultSet rs=null;
String sql="";
List<User>userList=null;
try{
    sql="select * from T_USER";
    conn=DBConnection.getConnection();
    prepareStatement=conn.prepareStatement(sql);
    rs=prepareStatement.executeQuery();
    userList=new ArrayList<User>();
    while(rs.next()){
```

```
            User user = new User( ) ;
            user. setId( rs. getString( " ID" ) ) ;
            user. setIcode( rs. getString( " ICODE" ) ) ;
            user. setName( rs. getString( " NAME" ) ) ;
            userList. add( user) ;
        }
    } catch( Exception e) {
            logger. errorT( " sql:" + sql + ". 查询用户表异常:" + e. getMessage( ) ) ;
    } finally {
            DBConnection. freeConnection( conn, prepareStatement, rs) ;
    }
            return userList;
    }
```

8）切换至 NWDI 的“Development Infrastructure”视图，右击 Java 工程，定义“Public Part”公共部分，并将外部需要访问的类添加到公共部分中，如图 8-82 所示。

9）在“MyComponents”中创建一个 EAR 项目，并添加 Java 工程的依赖。这是由于在 SAP NetWeaver 平台中不能直接发布 Java 工程，必须通过 EAR 项目进行打包并发布，如图 8-83 所示。

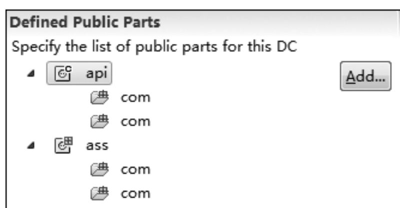


图 8-82　添加公共部分

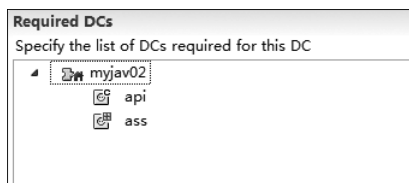


图 8-83　依赖工程明细 1

10）下面介绍如何在 Web Dynpro 中调用自定义数据库的这些连接方法并获取数据。首先创建一个新的 Web Dynpro 工程，并在 DI 视图中的“Dependencies”依赖中添加对于 EAR 工程的依赖，如图 8-84 所示。

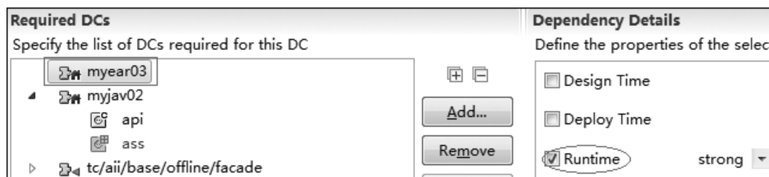


图 8-84　依赖工程明细 2

11）在 DI 视图中的“Dependencies”依赖中添加对于 Java 工程的依赖，如图 8-85 所示。

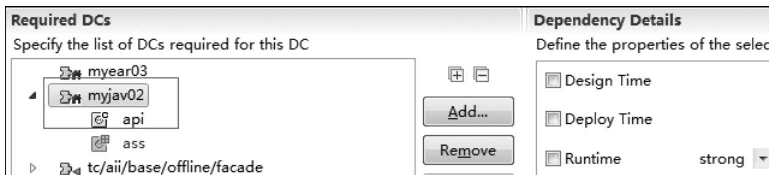


图 8-85　依赖工程明细 3

12）接着回到 Web Dynpro 视图，并创建对应的 Java Bean 模型，右击“Models”，并选择新建模型，在弹出的对话框中选择“Java Bean Model”，随后单击“Next”按钮，如图 8-86 所示。

13）选择 JavaBean 的源文件，选择来自于其他 DC 的公共部分，随后单击“Next”按钮，如图 8-87 所示。

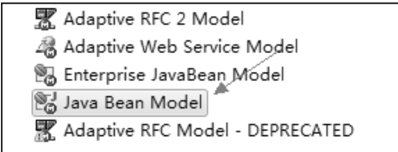


图 8-86 选择模型类型



图 8-87 选择 JavaBean 源文件类型

14）选中需要引用的公共部分，单击“Next”按钮，如图 8-88 所示。

15）选择需要使用的具体 Java 类，并单击“Add”按钮，如图 8-89 所示。

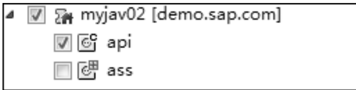


图 8-88 选择对应的公共部分



图 8-89 选择对应的类

16）完成所有配置后，单击“Finish”按钮，随后可以在项目目录中看到成功导入的模型，如图 8-90 所示。

17）接着在视图中创建对应属性的相关上下文，创建“User”结点和 3 个字符串类型的特性，以用于获取数据测试，如图 8-91 所示。

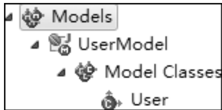


图 8-90 导入模型后的效果

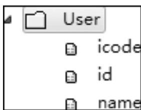


图 8-91 创建上下文清单

18）接着在视图中创建对应属性，右击“Outline”中的根结点，选择“Apply Template”，随后创建对应上下文的表格元素，如图 8-92 所示。

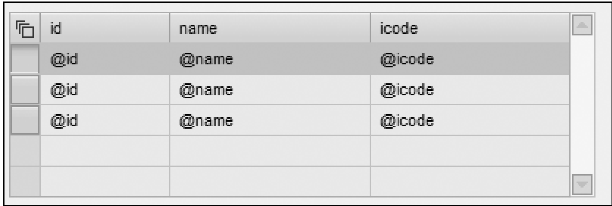


图 8-92 创建表格元素

19）进入 Web Dynpro 视图控制器的 Java 编辑器，在“wdDoInit”初始化方法中编写访问数据层的逻辑进行获取数据测试，添加如下代码：

```
List findUser = UserDao. findUser( );//访问 Dao
wdContext. nodeUser( ). invalidate( );//清空数据
for( int i = 0;i < findUser. size( );i += ) {
    IUserElement createAndAddUserElement = wdContext. nodeUser( )
    . createAndAddUserElement( );//Create Element
    User user = ( User) findUser. get( i);//赋值
    createAndAddUserElement. setId( user. getId( ) );
    createAndAddUserElement. setIcode( user. getIcode( ) );
    createAndAddUserElement. setName( user. getName( ) );
}
```

20）完成所有代码编写后，编译、部署相关工程（包括 Web Dynpro 项目、EAR 项目以及 Java 项目），运行应用程序后，就能看到对应的测试数据了，如图 8-93 所示。

id	name	icode
1	Gerny	1001

图 8-93　运行效果

8.6　Java 属性类的应用

Java 中有一个比较重要的类“Java. util. Properties”，主要用于读取 Java 的配置文件环境变量。配置文件是扩展名为“. properties”的文件，可以通过记事本或写字板直接进行编辑，也可以在 NWDS 中直接进行修改。文件中内容的格式是“键 = 值”的格式，文本注释信息则通过“#”来进行标记。

对于一般的 Java 项目或 J2EE 工程来说，在调整属性时直接修改“. properties”文件即可；但在 SAP NetWeaver 平台中，则是通过 NWA 平台进行属性的修改。下面通过一个实例来进行 SAP NetWeaver 平台中的属性类的应用，包括设计时的开发和运行时的调整两部分内容。

实例中配置的是服务器名和服务器端口两个属性，配置这两个属性的原因是很多应用程序在逻辑处理时需要访问服务器的功能，但由于 SAP NetWeaver 架构中的硬件配置，在应用从开发机传输至测试机、生产机的过程中经常需要变更服务器的链接信息，在使用 Java 属性类后，每次修改不再需要开发人员对代码进行维护，普通的运维管理人员就可在 NWA 端完成配置变更。

8.6.1　设计时配置

在开发过程中，将创建一个 EJB 工程，用于包含对应的属性文件以及读取该属性的公共方法。

开发实例：

1）打开 NWDS 开发者工作室，切换到“Java EE”透视图，如图 8-94 所示。

2）分别创建一个 EJB 和一个 EAR 工程，命名为“myejb04”和“myear05”，如图 8-95 所示。

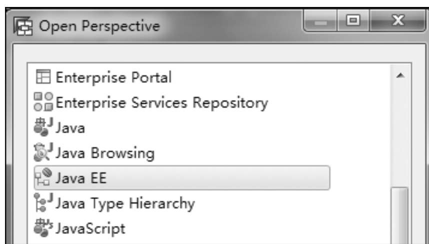


图 8-94　切换透视图

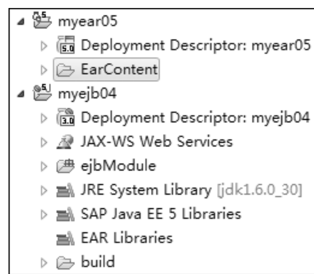


图 8-95　创建项目的目录结构

3）展开 EAR 工程，右击“META－INF”文件夹，并选择“New”，随后选择“File”，创建新文件，输入文件名“sap. application. global. properties”，如图 8-96 所示。

4）在文件中定义环境变量。本实例中定义了两个环境变量“SERVER_NAME”与“SERVER_PORT”，即服务器名和端口号，如图 8-97 所示。配置文件代码参考如下：

```
## SERVER_NAME
#? onlinemodifiable = true;
#% type = STRING;
SERVER_NAME = dev01. domain. com. cn

## SERVER_PORT
#? onlinemodifiable = true;
#% type = STRING;
SERVER_PORT = 50100
```

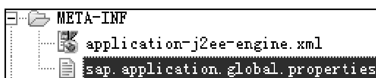


图 8-96　创建配置文件

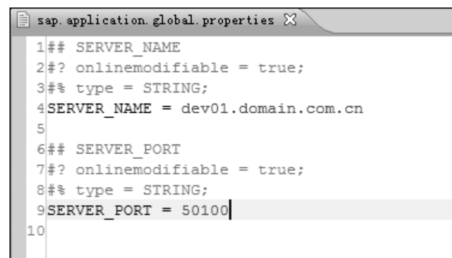


图 8-97　添加代码

在 EJB 中编写代码获取属性文件的配置参数值。首先新建一个 EJB，如图 8-98 所示。

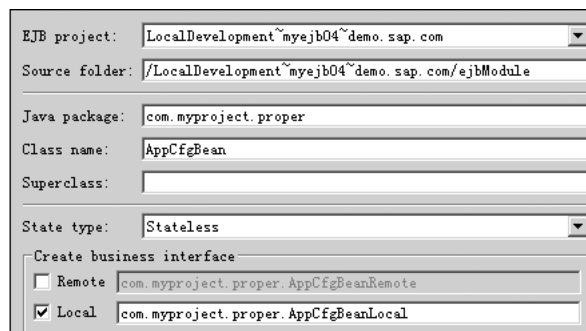


图 8-98　输入 EJB 信息

5）接着在接口中新定义获取属性方法，参考代码如下：

```
        getAppValue(String key);
@ Local
public interface AppCfgBeanLocal{
/ * *
 * 根据关键字从 properties 文件里获得变量值
 *
 * @ param key
 * @ return
 * /
public String getAppValue(String key);
}
```

6）在 Session Bean 中编写实现逻辑：

```
@ Override
public String getAppValue(String key){
        //TODO Auto - generated method stub
        String appValue = "";
        try{
                InitialContext ctx = new InitialContext();
                ApplicationPropertiesAccess appPropAcs = (ApplicationPropertiesAccess) ctx
                . lookup("ApplicationConfiguration");
                java. util. Properties appProp = appPropAcs
                . getApplicationProperties();
                if(null!= appProp &&!"". equals(key)){
                        appValue = (String) appProp. get(key);
                }
}catch(NamingException e){
        //系统日志
        //需添加 tc/je/appconfiguration/api 与 logging 的依赖
        logger. errorT("系统应用参数配置异常" + e. getMessage());
}
        return appValue;
}
```

7）编写工具类访问 EJB，添加类“com. myproject. util. AppPropUtil”：

```
/ * *
 * 根据关键字从 properties 文件里获得配置内容
 *
 * @ param key
 * 参数名称
 * @ return
 * /
public static String getAppConfByKey(String key){
        String value = "";
        try{
                InitialContext ctx = new InitialContext();
                                String    jndi    =    "    demo. sap. com/myear05/LOCAL/AppCfgBean/
com. myproject. proper. AppCfgBeanLocal";
                AppCfgBeanLocal local = (AppCfgBeanLocal) ctx. lookup(jndi);
                value = local. getAppValue(key);
        }catch(NamingException e){
```

```
            //记录系统日志
            logger.errorT("系统应用参数配置出现异常" + e.getMessage());
        }
        return value;
    }
```

8）添加 Util 类至公共部分，以便其他应用程序进行公共方法的调用，如图 8-99 所示。

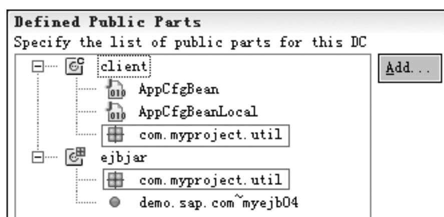


图 8-99　定义公共部分

9）编译、部署 EAR 项目后，在 Web Dynpro 项目中就可以通过 API 的使用获取属性文件中的值，Web Dynpro 参考代码如下：

```
//get system properties server_name
String appConfByKey = AppPropUtil.getAppConfByKey("SERVER_NAME");
wdComponentAPI.getMessageManager().reportSuccess(
"SERVER_NAME:" + appConfByKey);
```

10）运行 Web Dynpro 应用程序后，就能打印出获取的属性值，如图 8-100 所示。

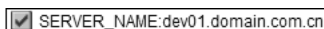


图 8-100　提示消息

8.6.2　运行时调整

将相应的属性文件部署至 Java 应用服务器后，当需要修改时，可直接通过 NWA 对配置文件进行修改和更新。

开发实例：

1）使用管理员账户登录 NWA 平台，选择“Configuration”→“Infrastructure”，单击“Java System Properties”链接，如图 8-101 所示。

2）搜索对应的 EAR 名称，支持模糊查询，如图 8-102 所示。

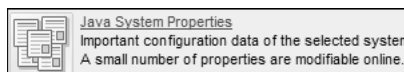


图 8-101　单击链接

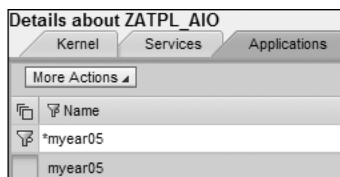


图 8-102　查询对应的应用程序

3）选择对应的 EAR 项目后，在下方的属性页签中就能看到对应的属性名称，如图 8-103 所示。

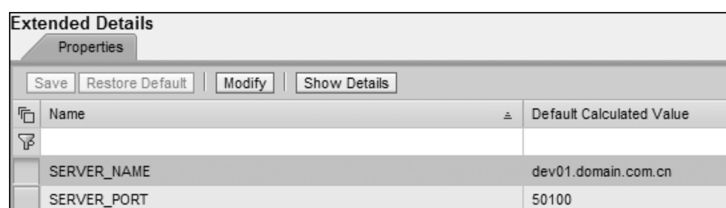


图 8-103　找到配置文件的属性

4）单击选中“SERVER_NAME”，单击“Modify”按钮，在弹出的对话框中输入需要修改的新值“dev02. domain. com. cn”，随后单击“Set”按钮，如图 8-104 所示。

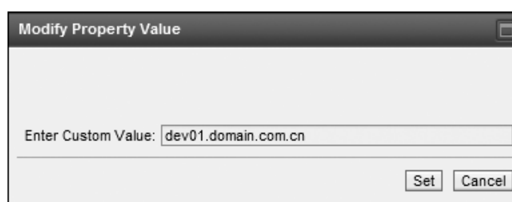


图 8-104　修改属性值的对话框

5）对话框设置完成后，还需要单击“Save”按钮保存修改，如图 8-105 所示。

Default Calculated Value	Modifiable Online	Custom Calculated Value
dev01.domain.com.cn	☑	dev02.domain.com.cn

图 8-105　修改后的效果

6）再次运行 Web Dynpro 测试程序，打印出更新后的属性文件的对应值，如图 8-106 所示。



图 8-106　提示消息

8.7　日志管理

在实际的项目应用中，日志的记录以及跟踪非常重要，尤其是紧急性的排错等问题对于日志的依赖更是很大，因此需要掌握程序开发过程中日志记录的方法，并且规范地记录 Web Dynpro 应用程序中或是 BPM 业务流程中关键动作或步骤产生的日志。

8.7.1　设计时日志记录

对于新建的 Web Dynpro 项目，在切换到 DI 透视图后，可以在默认的“Dependencies”项目依赖的选项卡中发现已默认添加了对“tc/bl/logging/api”标准开发组件的依赖。该开发组件包含了日志记录的公共类及相关方法，如图 8-107 所示。

在 Web Dynpro 的任意一个视图中，也能找到默认的日志记录的实例“logger”，如图 8-108 所示。

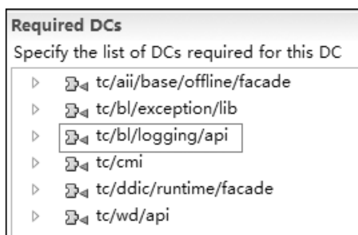


图 8-107　依赖项目清单

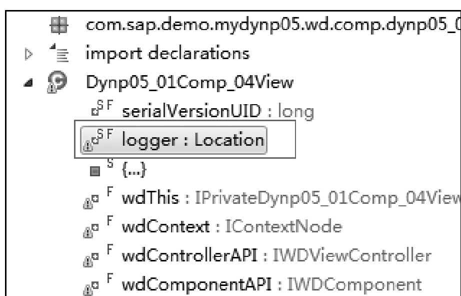


图 8-108　默认的日志记录类

默认的定义代码，如图 8-109 所示。

```
/**
 * Logging location.
 */
@SuppressWarnings("unused")
private static final com.sap.tc.logging.Location logger =
  com.sap.tc.logging.Location.getLocation(Dynp05_01Comp.class);
```

图 8-109　标准定义代码

在这些标准类和参数的支持下，在 Web Dynpro 的视图控制器或者组件控制器中，都可以直接使用标准的日志类来进行运行日志的记录。根据日志级别的高低，logger 类分别提供了不同的记录方法，如图 8-110 所示。

```
logger.warningT("操作记录警告!");//记录 warning 级别的日志
logger.infoT("操作记录成功!");//记录 info 级别的日志
logger.errorT("操作记录失败!");//记录 error 级别的日志
```

```
public void onActionLog(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionLog(ServerEvent)
      logger.warningT("操作记录警告!");// 记录warning级别的日志
      logger.infoT("操作记录成功!");// 记录info级别的日志
      logger.errorT("操作记录失败!");// 记录error级别的日志

  //@@end
}
```

图 8-110　不同类型日志记录示例

需要注意的是，在使用以上代码时，针对 SAP 的 Web Dynpro 项目会默认添加日志组件的依赖，而对于其他类型的项目，则需要手工添加对于日志类的依赖，并初始化 logger 实例，之后才能通过“warningT”等方法进行日志记录。

8.7.2　运行时日志查看

对于应用程序运行时的日志查看，NWA 中提供了统一的查看日志视图，进入 NWA 平台，找到“Troubleshooting”→“Logs and Traces”→“Log Viewer”，如图 8-111 所示。

随后在菜单中选择“View”→“Open View”→“Developer Traces”，如图 8-112 所示。

刷新能够不断看到最新的日志，可以使用“Advanced Filter”高级过滤器对日志做进一步

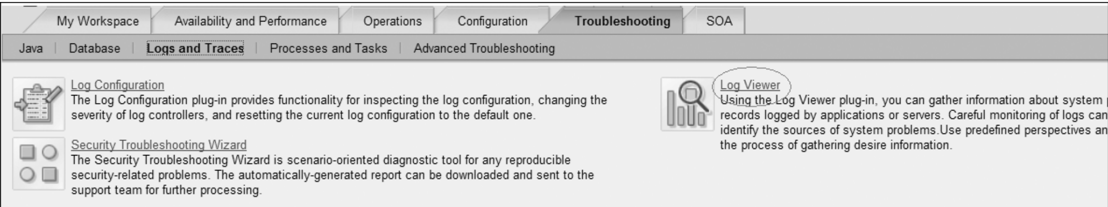


图 8-111　NWA 平台中的日志查看器

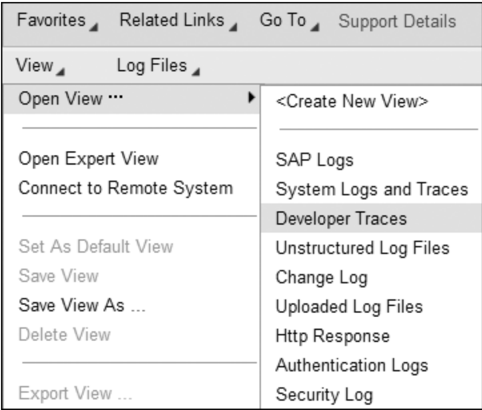


图 8-112　查看开发日志的菜单选择

地筛选与过滤，也可以在表格最上放的输入框中输入关键字进行模糊查询，如图 8-113 所示。

Time	Message
15:32:56:389	操作记录失败!

图 8-113　查看新日志记录

注：在 Web Dynpro 测试中，同时记录了消息、警告和错误，但由于默认的日志级别设置，因此此处只能看到错误消息。

8.7.3　日志级别配置

由于不同系统的体量不同，以及需要记录的日志具体要求也不尽相同，所以需要针对不同的情况进行个性化的日志级别配置，以此优化日志使用效率和服务器硬盘容量负载。

在实际项目开发中，对于日志记录最低的要求是记录错误级别的日志，特别是对于应用程序异常或者接口功能报错等都需要有对应的后台日志，以便进行追踪和定位；同时对于代码的调试或者功能测试，也经常需要将操作日志，即消息级别的日志记录下来。

由于实际的生产环境，往往会有大量的业务数据运行，这期间也会产生庞大的日志文件，服务器对日志文件的频繁操作会大大影响其性能，所以建议对于生产环境采用默认的错误级别，而在开发或测试环境中，可以调整配置为信息级别，以提供对开发人员更好的支持。

配置实例：

1）进入 NWA 平台，找到“Troubleshooting”→“Logs and Traces”→“Log Configuration”，如图 8-114 所示。



图 8-114　NWA 平台中的日志配置

2）选择“Tracing Locations”并输入确切的日志记录位置，如图 8-115 所示。

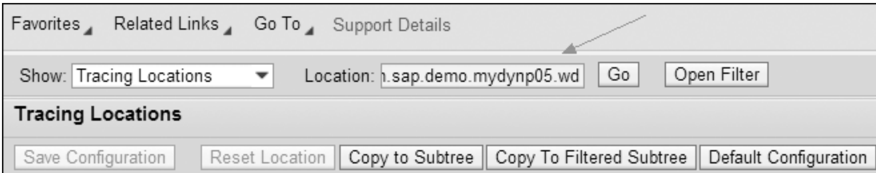


图 8-115　日志记录位置的操作界面

3）单击“Go”按钮，在弹出的对话框中选择项目的根目录，单击“OK”按钮，如图 8-116 所示。

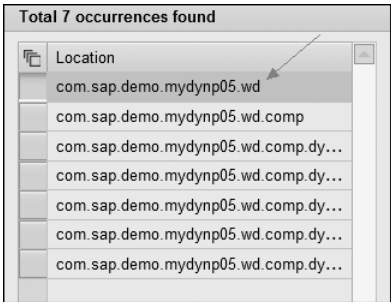


图 8-116　选择对应目录

4）在默认情况下，对应目录的日志级别是“Error”，即错误级别。对它进行重新设置，可以显示级别较低的开发日志，如图 8-117 所示。

demo	Error	Inherited From Parent
mydynp05	Error	Persisted Configuration
wd	Error	Inherited From Parent
comp	Error	Inherited From Parent

图 8-117　查看日志配置级别

5）将“wd”目录的日志级别修改为“Warning”，即警告级别，单击“Copy to Subtree”应用到所有子目录中，随后单击“Save”对配置进行保存，如图 8-118 所示。

demo	Error	Inherited From Parent
mydynp05	Error	Persisted Configuration
wd	Warning	Persisted Configuration
comp	Warning	Inherited From Parent

图 8-118　修改日志配置级别

6）再次运行 Web Dynpro 程序后，刷新日志视图，同时显示了错误和警告，如图 8-119 所示。

Time	Message
15:46:03:438	操作记录失败!
15:46:03:437	操作记录警告!

图 8-119　查看日志记录

7）再次修改目录的日志级别，选择“Info”消息级别并保存，如图 8-120 所示。

demo	Error	Inherited From Parent
mydynp05	Error	Persisted Configuration
wd	Info	Persisted Configuration
comp	Info	Inherited From Parent

图 8-120　修改日志配置级别

8）再次运行程序并刷新日志视图，即可同时显示 3 种级别的日志，如图 8-121 所示。

Time	Message
15:49:22:437	操作记录失败!
15:49:22:437	操作记录成功!
15:49:22:436	操作记录警告!

图 8-121　查看日志记录

第 9 章　开发基础架构

在了解 SAP 的开发基础架构之前，先来看下一些传统项目的团队 Java 开发状况：团队在开发时需要手动地更新 Java 类库，在本地开发并完成代码编写后手动地将项目发布至测试服务器进行测试运行；在多人协作时容易出现类库不一致、不同开发人员负责的功能代码难以融合等问题；团队在各自提交代码进行程序更新或升级时没有系统或平台支持，运维人员难以对开发人员独立的部署更新进行有效地管理和监控。比较常见的问题如下：

1）在开发阶段没有同步最新的代码进行使用。

2）在匹配外部 Java 类库时未使用最新版本，或多个开发成员使用的版本不一致。

3）由于没有完整的载入，所有的必要依赖工程导致本地编译出错。

4）多个开发人员修改同一功能导致代码冲突。

5）本地编译与服务器端编译结果不一致。

6）开发工程环境不一致，或测试服务器配置差异，导致运行结果不一致。

7）定位问题、解决文件冲突、同步代码和类库等工作过程冗长枯燥。

为了解决这一系列的问题，提升开发团队的工作效率，SAP 推出了 SAP NetWeaver 开发基础架构（SAP NetWeaver Development Infrastructure，NWDI），它提供给了开发人员尤其是开发团队一个强大的集成化开发平台。它可以覆盖整个项目的开发周期，涵盖了从设计、开发、编译、发布、同步到更新等内容范围。由于代码的集成管理以及开发环境的高度统一，避免了各自单独开发时的同步问题和更新问题，加速了团队整体的开发进度；平台的运行环境已集成在 Java 开发人员熟悉的，以 Eclipse 为核心的开发工具中，并支持各种各样的 SAP NetWeaver 平台的 Java 相关项目创建、编译以及部署服务器工作；可靠的传输同步机制，并且有强大的 SAP NetWeaver 平台对传输请求以及代码版本进行统一的监控和管理。

如果接触过 SAP ABAP 开发，在使用了 NWDI 以后，一定会有一种似曾相识的感觉。从某种意义上来说，SAP 正是想要将 ABAP 开发中的一整套管理机制搬到它想要涉足的 Java 世界中。例如，传输请求、“Check in” 和 “Check out” 动作、冲突代码整合、代码的版本管理和比较、传输管理和监控，这些 NWDI 中的种种功能特性其实都能在 SAP 的 ERP 系统中找到一些可以追寻的踪影，见表 9-1。

表 9-1　SAP 中 Java 与 ABAP 开发的相似之处

不同开发阶段	SAP 中的 Java 开发	SAP 中的 ABAP 开发
开发环境准备	登录 NWDI 的开发控件	登录 SAP 开发系统（如 ECC）
面向团队开发	复制、下载源代码，进行相关开发工作	修改并同时锁定开发对象
代码调试	基于 NWDI 服务端环境进行编译，激活更改	激活代码修改
集成测试	与 NWDI 架构一致的环境中测试应用程序	在 SAP 系统中直接进行代码或应用运行测试
统一运行	释放请求并进行传输	释放请求并进行传输

9.1 NWDI 的组成

在 NWDI 整个架构中，包含了 DTR、CBS、CMS、SLD 和 NWDS 等几个部分。

- DTR（Design Time Repository，设计时资源库）用于存储所有的代码文件对象。
- CBS（Component Build Service，组件编译服务）用于存放服务器端开发环境的所有类库，也用于编译和存放所有新创建的开发组件。CBS 与 DTR 紧密集成在 NWDI 中。
- CMS（Change Management Service，变更管理服务）用于对所有开发对象的变更管理。
- SLD（System Landscape Directory，系统场景目录）用于定义源代码运行环境的服务器版本、所用到的具体产品、所用到的具体软件组件。严格来说，SLD 并不属于 NWDI，但是 NWDI 的运行依赖于 SLD 的环境配置。
- NWDS 即为 SAP NetWeaver 开发者工作室，相关介绍可参考本书第 1 章内容。

SAP 中 Java 和 ABAP 开发的环境比较见表 9-2。

表 9-2　SAP 中 Java 和 ABAP 开发的环境比较

场景和功能	SAP 中的 ABAP 开发	SAP 中的 Java 开发
编辑器	系统中的对象导航器（SE80）	SAP NetWeaver 开发者工作室（NWDS）
编译器	系统核心功能	NWDI 中的 CBS 服务
运行时环境	ERP 服务器实例	SAP NetWeaver 服务器实例
代码版本管理	系统中的版本管理	NWDI 中的 DTR 资源库
传输机制	传输管理系统（STMS）	NWDI 中的 CMS 服务

由表 9-2 可以看出，SAP 迫切想要通过 NWDI 将 ABAP 开发中成熟的管理模式和成功的开发经验复制到 Java 开发中。下面介绍整个 NWDI 的布局架构将如何覆盖整个 Java 项目的开发生命周期，如图 9-1 所示。图 9-1 的左侧是开发人员个人计算机中的 NWDS 开发环境，即开发人员使用的本地开发工具，右侧是 NWDI 的开发者架构（包含了 DTR、CBS、CMS 等功能），下方是 SAP NetWeaver 中 SLD 的系统目录，这些专业化的工具保证了开发人员在 SAP 的 Java 开发中的高效和便捷。

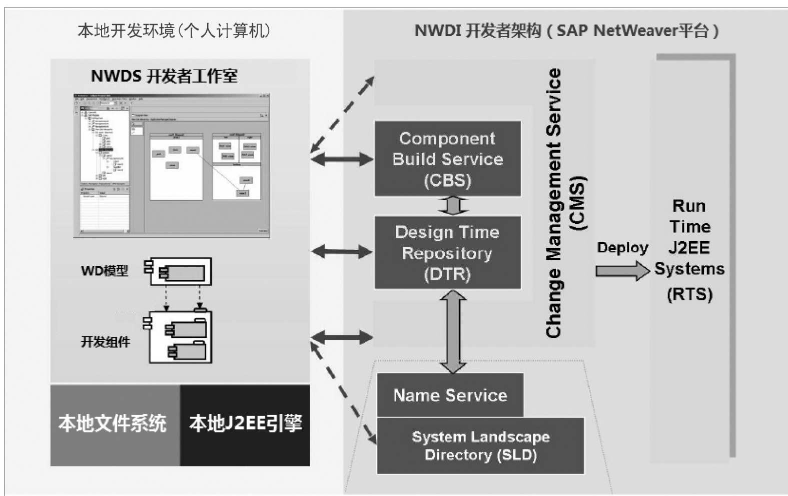


图 9-1　NWDI 架构与本地开发环境的协作

9.1.1 设计时资源库

DTR 提供了一个对所有源代码进行集中存储、管理以及版本控制的资源库，同时还针对所有的开发对象进行自动的代码冲突检测和处理。作为客户端的 DTR 部分深度集成在 SAP NetWeaver 开发者工作室（NWDS）中，提供了对所有可用项目类型的支持。

对于大规模的开发项目，DTR 提供了分布式的同步开发机制：

- 不同的开发小组可以同步进行开发工作，即使不同开发人员分布在不同的工作场所甚至不同城市，他们也可以通过与服务端 DTR 的连接来保证所有的修改能够顺利进行。
- DTR 会自动检测同一个开发对象是否会产生两个不同的版本，并在修改之前就进行提醒，以防止源代码不一致的情况出现。
- 当代码冲突在 DTR 中出现时，也可以通过 DTR 进行处理并解决冲突，DTR 可以提供开发对象的版本历史、不同版本的代码比较以及用户处理的推荐选项，方便不同开发人员处理不同情况的代码冲突。

DTR 对于开发对象的文件和文件目录管理提供了以下功能：

- 虽然所有开发对象都是存储在 DTR 的服务器端，但在开发人员本地开发环境中，通过 SAP NetWeaver 开发者工作室登录 NWDI 后，就能方便地查看所有相关的文件和目录，并根据实际需求进行检出和修改。
- 源代码文件会根据不同的开发语言分别进行存储，但所有开发对象的文件都可以进行存储管理和版本管理。

DTR 还提供了“检入/检出”（Check In/Check Out）模型进行开发对象管理：

- 开发人员要修改一个开发对象，首先要通过“Check Out”检出动作获取最新版本的代码文件拷贝，随后在本地的代码文件或开发对象中进行相关修改，在完成开发后，通过“Check In”检入动作将本地修改的开发对象同步至服务端。
- 而每次开发人员对于开发对象的修改，都会产生一个活动（Activity），在活动中记录了所有本地修改的开发对象，以及开发对象中具体修改的内容。

对于开发人员来说，DTR 包含了客户端和服务端，服务端一般由项目管理员或开发组长进行统一配置，可以通过浏览器地址直接进行访问：http://<host>:<port>/devinf；而客户端就集成在 SAP NetWeaver 开发者工作室中，可以在视图选择中找到对应的“Design Time Repository”视图。需要注意的是，DTR 不仅仅只是一个开发对象的共享目录，它还与 CBS 和 CMS 等 NWDI 中的服务紧密集成，共同为 SAP 的 Java 开发人员提供了良好的开发体验。如果需访问服务器端 DTR，则可通过浏览器访问“http://<host>:<port>/devinf”并找到 DTR 的对应链接，如图 9-2 所示。

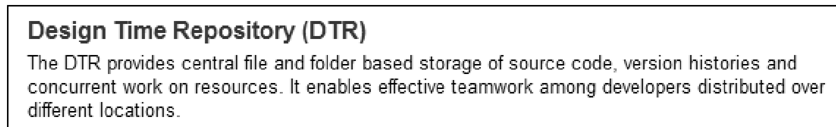


图 9-2　服务器端的 DTR 链接

进入 DTR 后，首先看到的就是所有开发对象的资源库浏览器，如图 9-3 所示。

在上方系统工具中可以找到“reports”报表工具，支持对不同类型 DI 活动的搜索、版

本冲突、源文件路径、源文件的查询以及工作空间的对比与整合等，如图 9-4 所示。

Repository Browser

Resource	Author	Content Type
/		
system-tools		
ws	System	

图 9-3　资源库浏览器

- Activity Search
- Conflict Search
- File/Folder Search
- Resource Lookup
- VersionSet Comparison
- Workspace Comparison
- Workspace Integrations

图 9-4　查看具体开发对象

在展开对应的目录后，就能看到所有开发项目在 DTR 中存储的目录结构以及对应的开发对象，右侧还提供了一些活动处理按钮对开发对象进行简单处理，如图 9-5 所示。

Resource	Last Modified	Actions
/		
ws		
BPMD		
cn.com_BPMS		
dev		
inactive		
DCs		
myproject.com		
pjct		
base		
my01_wd		

图 9-5　不同目录以及处理界面

9.1.2　组件编译服务

CBS 是基于 J2EE 架构的服务端的数据库应用程序，它基于组件模型架构并且专门用于编译 SAP 开发组件。它既可以用于开发阶段的编译工作，也可以用于最终代码整合时的工具，最重要的是，它与 DTR 资源库紧密集成。要了解 CBS 首先要知道它的一些特性：

- 存储软件组件不同开发状态的编译版本信息。
- 根据开发人员的需求可以对代码修改进行即时的编译，由于客户端环境的存在，不再像过去那样需要将代码同步至一个完整的编译环境中运行，完全可以在本地开发工具中编译通过以后直接更新。
- 自动重编译开发对象所依赖的开发组件，保证开发人员在检出代码时就能得到与服务器端同步的相关运行环境。
- 提供了网页端管理页面，方便管理员进行维护和监控。
- 允许不同编译请求同步执行，提升了整体的编译效率。

当 CBS 和 DTR 放在一起时，需要学习一个新的概念——“激活”（activation）。激活是开发对象从 DTR 进行检入以后的一步额外的动作，作用是使所有的修改生效。举例来说，一个功能模块由不同开发人员同时进行开发修改，当完成一部分功能时还无法顺序运行程序，所以完成工作的开发人员需要先进行检入的动作，而在所有开发修改都完成并成功检入

后，才需要将所有的修改进行启用，也就是“激活”动作使修改生效。另外，激活的过程并不需要额外的编译。

为什么在了解“激活”时需要捎带上之前介绍的 DTR 呢？这是因为在客户端的 DTR 中包含了两个不同的工作空间：未激活工作空间，所有的开发对象修改都在这里进行；已激活工作空间，这里所有的源代码都可以被安全使用，因为它们在检入前都已经过了 CBS 的编译验证。其中，已激活工作空间会随时与 CBS 的编译空间中的可用档案进行同步，以保证开发对象的可用性。

CBS 的编译空间位于 CBS 服务器上，每个服务器可以有多个编译空间且相互独立，编译空间用于存放软件组件在不同开发状态的编译档案，而其中最新的可用档案会与 DTR 中已激活工作空间的开发对象进行同步。

与 DTR 类似，CBS 也同样有客户端的激活视图，已集成在 SAP NetWeaver 开发者工作室中，用于激活已成功检入的开发对象的修改内容。当然激活的对象就是在 DTR 中进行修改时生成的活动（Activity）。另外，CBS 还包括了服务器端的 CBS 服务器，用于管理和监控 CBS 的编译空间。如果需访问服务器端 CBS，则可通过浏览器访问“http://<host>:<port>/devinf”并找到 CBS 的对应链接，如图 9-6 所示。

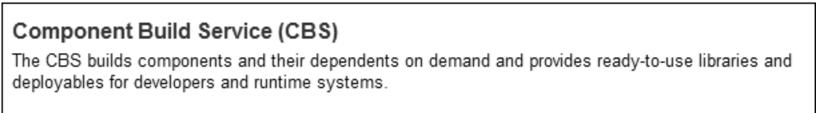


图 9-6　服务器端的 CBS

1）在进入 CBS 页面后，首先看到的是 CBS 的编译空间（buildspaces），如图 9-7 所示。

Buildspaces

View | Edit | Delete | Refresh

Name	Queued Request	Processing Request	Failed Requests (Last Week)	Total DCs	Broken DCs
MPD_BPMD_C	0	0	0	1,535	0
MPD_BPMD_D	0	1	0	1,535	0

图 9-7　编译空间

2）单击不同的编译空间链接，就能看到对应的不同内容（Compartments），如图 9-8 所示。

Compartments

Buildspace: MPD_BPMD_D | Build... | Initialize | Open in New Window | Refresh

Compartment	SC Vendor	SC Name	Archives	Sources	Activation Possible
cn.com_BPMS_1	cn.com	BPMS	–	✓	✓

图 9-8　编译空间中的不同内容

3）在开发组件页签中，可以看到当前空间包含的所有软件组件和开发组件，如图 9-9 所示。

4）可以通过单击右侧的“Pending Activities”对应的链接，查看未激活的活动内容，如

图 9-10所示。

Development Components | DTR Workspaces

DC Filter: All DCs　Build Variant:　Refresh

All Development Components

DC Name	DC Vendor	Collisions	Compile State	Metadata	Cycle-Free	Used DCs
pjct/base/my01_wd	myproject.com	View				
pjct/base/my02_wd	myproject.com	View				

图 9-9　查看软件组件和开发组件

DC Vendor	Collisions	Compile State	Metadata	Cycle-Free	Used DCs	Pending Activities
myproject.com	View					View
myproject.com	View					View

图 9-10　查看活动内容的操作界面

5）在查看活动内容的界面，可以对清单和开发对象进行模糊查询，如图 9-11 所示。

Activities

Buildspace: MPD_BPMD_D

Compartment: cn.com_BPMS_1

Simple Search | Advanced Search | Pending Activities | Activated Activities

Owner: *

Max. No. of Hits *: 1,000

Development Component: myproject.com/pjct/base/my02_wd　Reset

Search

Search Results

Activate | Activate even if build fails

Name
modify my02_wd My02CompView by zhaox at date1449

图 9-11　查看活动内容的界面

6）单击表格上方的“Activate”按钮即可通过服务器端 CBS 激活对应的活动，CBS 中也会自动生成请求号（与 NWDS 中操作的效果相同），并通过弹出的请求窗口查看激活请求的结果，激活请求执行成功，如图 9-12 所示。

Requests

Buildspace: MPD_BPMD_D　Buildspace-Independent Search

Simple Search | Advanced Search | Open Requests | Completed (Unconfirmed) Requests

Request ID: 13,270

Search

Search Results

Confirm | Cancel | Suspend | Resume

Request ID	Request State	Request Type	Owner	Confirmed	Request Log
13,270	SUCCEEDED	ACTIVATE	dev_zhaox	✓	View

图 9-12　查看请求状态

9.1.3　变更管理服务

在 NWDI 中，完整的开发场景是由一个或多个 CBS、DTR 服务器以及一个 SLD 对应的命名服务器组成的，而 CMS 就是负责整个开发场景定义以及管理的中央服务。

CMS 主要用于配置开发前景，并且控制所有开发对象的系统传输及应用部署。另外，CMS 还负责对 CBS 创建开发对象的归档，导出对应 SCA 文件并进行系统间版本传输。CBS 按时间顺序对不同激活动作产生不同的归档文件，CMS 保留了历史版本的 archive，并确保了非开发级别的运行时部署。同时 CMS 可以自动对开发轨道配置以及软件组件的依赖同步地进行修改。

在 CMS 中可以看到轨道的清单（Track List），如图 9-13 所示。

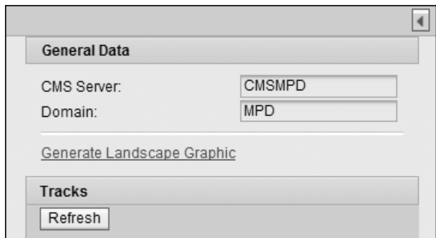


图 9-13　CMS 中的轨道清单

另外，也可以查看具体的运行时系统和明细信息，如图 9-14 所示。

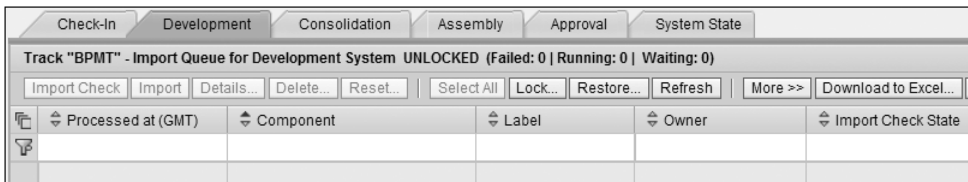


图 9-14　开发组件的操作界面

开发实例：

1）在客户端 NWDS 中，打开“Transport View”，就可以对修改完成的动作进行释放，如图 9-15 所示。

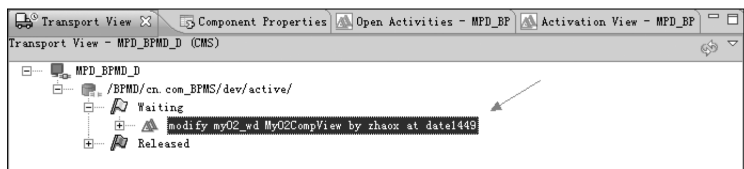


图 9-15　查看完成的动作

2）右击需要释放的动作，随后选择“Release”进行释放，如图 9-16 所示。



图 9-16　释放动作的菜单选择

3）在弹出的对话框中输入变更请求的描述，确认动作选择无误后，单击“Release activities”按钮即可开始释放，如图 9-17 所示。

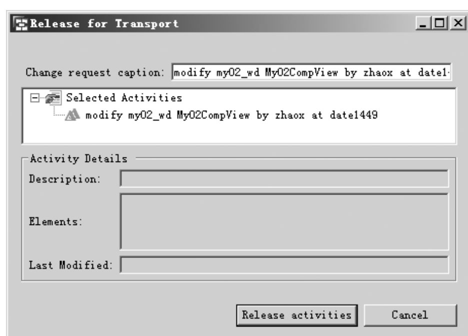


图 9-17　确认对话框

4）释放后，在传输视图中可以看到对应的内容清单，如图 9-18 所示。



图 9-18　传输内容

5）接着进入服务器端的 CMS，选择“Runtime Systems”→“Consolidation”，在清单中就能看到刚才释放的动作，对应的状态是“等待导入”，如图 9-19 所示。

Check-In | Development | Consolidation | Assembly | Approval | System State

Track "BPMD" - Import Queue for Consolidation System UNLOCKED (Failed: 0 | Running: 0 | Waiting: 4)

Import Check | Import | Details... | Delete... | Reset... | Select All | Lock... | Restore... | Refresh | More >> | Download to Excel... | Export SCA... | Deployment... | Go to History

Processed at (GMT)	Component	Label	Owner	Import Check State	State
2014-01-11 07:17:18	cn.com_BPMS	modify my02_wd My02CompView by zhaox ...	dev_zhaox	Not executed	Waiting for import

图 9-19　轨道操作界面

6）选中该行后，单击上方的“Import”按钮进行导入。导入成功后状态更新成为了“Import finished”，如图 9-20 所示。

Processed at (GMT)	Component	Label	Owner	Import Check State	State
2014-01-11 07:24:58	cn.com_BPMS	modify my02_wd My02CompView by zhaox ...	dev_zhaox	Not executed	Import finished

图 9-20　查看导入状态

7）接着进入“Assembly”页签，之前导入的动作状态变成了“Waiting for assembly”，如图 9-21 所示。

Check-In | Development | Consolidation | Assembly | Approval | System State

Track "BPMD" - Component Versions for Assembly (Failed: 0 | Running: 0 | Waiting: 1)

Assemble Component(s)... | Details... | Reset... | Select All | Refresh | Forward from History... | Attach from History... | Download to Excel... | Export SCA... | Go to History

Processed at (GMT)	Component	Release	SP Number	Patch Name	Patch Level	New	State
2014-01-11 06:43:30	cn.com_BPMS	1.0	1	BPMD.01110642	1	1 / 0	Waiting for assembly

图 9-21　轨道操作界面

8）选中该动作，单击上方的“Assemble Component(s)”按钮，在弹出的对话框中设置对应的属性和补丁等级，单击“Assemble”按钮开始集成，如图 9-22 所示。

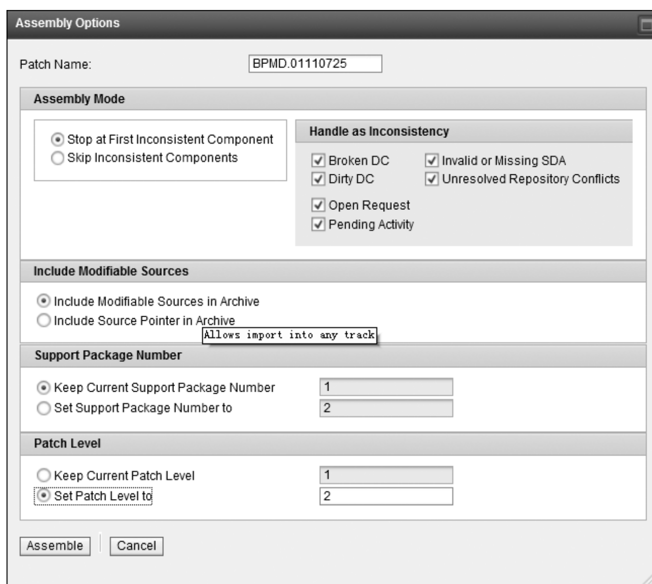


图 9-22　集成对话框

9）在集成完毕后，就可以单击“Export SCA”按钮，将对应开发对象进行导出，如图 9-23所示。

Track "BPMD" - Component Versions for Assembly (Failed: 0 | Running: 0 | Waiting: 0)

Assemble Component(s)... | Details... | Reset... | Select All | Refresh | Forward from History... | Attach from History... | Download to Excel... | Export SCA... | Go to History

Processed at (GMT)	Component	Release	SP Number	Patch Name	Patch Level	New	State
2014-01-11 07:32:01	cn.com_BPMS	1.0	1	BPMD.01110725	2	0 / 0	assembled

图 9-23　轨道操作界面

10）选择相关联的标准软件组件，单击“Create Archive(s)”按钮，如图 9-24 所示。

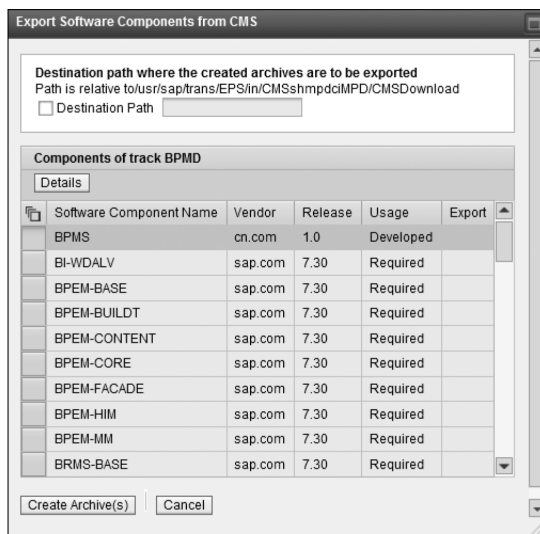


图 9-24　创建归档对话框

11）归档完成后，在清单中就能看到下载的链接，单击“download”即可开始下载，如图 9-25 所示。

Components of track BPMD

Details

Software Component Name	Vendor	Release	Usage	Export
BPMS	cn.com	1.0	Developed	Download

图 9-25　下载操作界面

12）使用本地的 SCA 文件进行导入和部署到新的服务器端，如图 9-26 所示。



图 9-26　通过 NWDS 进行导入

注： 部署时 Settings 选择只更新低版本 DC.

9.2　NWDI 初始化配置

在开始使用 NWDI 之前首先要确认已正确安装相关组件，在安装 SAP NetWeaver 时也必须勾选 NWDI 相关的内容条目。NWDI 的初始化配置有两种方式，在 NW 7.2 之前一直是采用手工配置的方式，但由于手工配置步骤繁冗并且纠错能力较低，NW 7.2 之后推出了自动配置向导执行。这里介绍通过自动向导进行配置的方式，可以有效避免配置过程中出现遗漏。

9.2.1　CMS 配置向导

自动向导是 NWA 提供的一个集成化配置工具，通过浏览器地址直接访问：http://<host>:<port>/devinf，随后找到“CMS Wizards” CMS 向导链接。

1）在页面中“CMS Wizards”下方选择“Create an Application Skeleton”，创建一个新的应用框架，随即进入具体的配置向导中，如图 9-27 所示。

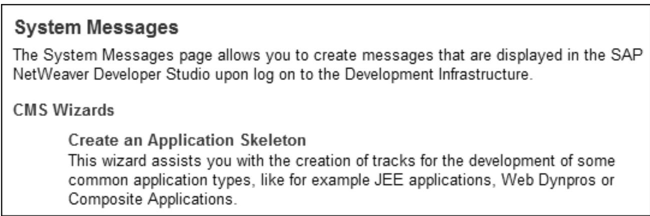


图 9-27　服务器端 CMS 向导

2）填写 CMS 相应的基本参数和连接信息，填写完成后单击“Next”按钮，如图 9-28 所示。

图 9-28　配置信息

3）创建具体的轨道，即开发架构的实例，输入标识和描述即可，如图 9-29所示。

图 9-29　配置轨道

4）随后输入默认的软件组件的名称、供应商和版本号，如图 9-30 所示。

图 9-30　输入组件信息

5）信息确认完毕后，向导将自动开始创建 DI 轨道、软件组件以及往 CMS 导入相关目

录和归档文件的动作，并且接下来的过程不需要其他的参数输入，用户可以从 CMS WebUI 中查看执行结果并做执行监控，如图 9-31 所示。

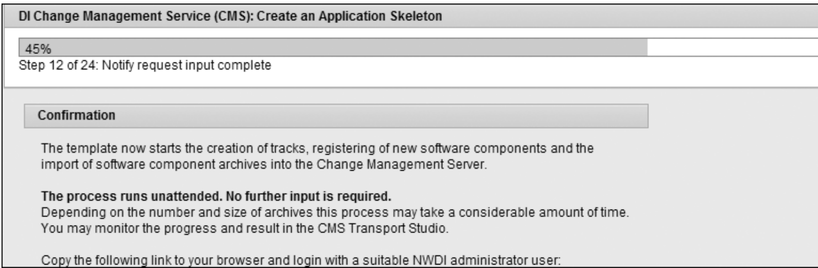


图 9-31　自动执行过程

6）进入 CMS 监控器，可以看到向导自动新建的轨道实例“BPMD”，如图 9-32 所示。

7）由于需要导入 NWDI 相关的所有标准包（即软件组件），因此向导会在最后一步导入软件组件包时运行一段时间，需要耐心等待，如图 9-33 所示。

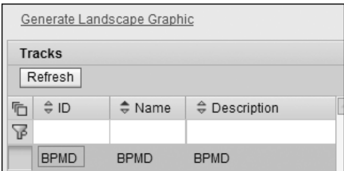


图 9-32　创建完成的轨道

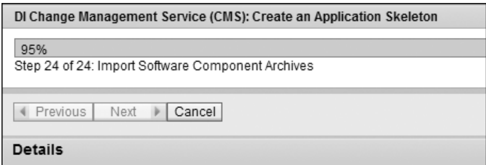


图 9-33　自动执行过程

8）向导执行完毕，可以看到对应的提示消息，如图 9-34 所示。



图 9-34　执行成功后的提示消息

9.2.2　SLD 命名保留

在创建 DI 轨道之后，还需要通过 SLD 命名保留，建立软件组件（SC）的供方名称和开发组件（DC）的默认命名前缀。通过创建的 SC 与 DC 的命名前缀，规范地定义了 DTR 中的路径名，方便后期的开发对象管理和维护。

配置实例：

1）通过浏览器地址直接访问：http://<host>:<port>/sld，单击“Name Reservation”进入详细配置界面，如图 9-35 所示。

2）选中“Name Prefixes”页签，在“SC Name”中选择“New Name Prefix”，创建一个新的命名前缀，在弹出的对话框中输入相关必要信息，保存所有配置，如图 9-36 所示。

3）保存修改后，就可以在表格中查看已创建的 SC 命名前缀了，如图 9-37 所示。

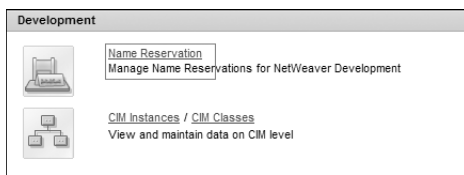


图 9-35　服务器端命名保留链接

Define a new name prefix for name reservations on this server
Quantity: ⦿ Single ○ Bundle
Name Category: * Software Component Name
Name Prefix: * myproject.com
Purpose: * Used by developer
Owner: dev_zhaox
Description:

图 9-36　输入详细信息

Name Prefixes | Names
Name Category: Software Component Name (2) Filter by Name: Go
New Name Prefix... | Remove | Refresh

Name Prefix	Owner
myproject.com	dev_zhaox

图 9-37　创建前缀名操作界面

4）将“Name Category”切换到开发组件名称，并创建一个命名前缀，输入必要信息并保存，如图 9-38 所示。

Define a new name prefix for name reservations on this server
Quantity: ⦿ Single ○ Bundle
Name Category: * Development Component Name
Name Prefix: * myproject.com/pjct
Purpose: * Used by developer
Owner: dev_zhaox
Description:

图 9-38　输入详细信息

5）保存修改后，就可以在表格中查看已创建的 DC 命名前缀了，如图 9-39 所示。

Name Prefixes | Names
Name Category: Development Component Name (2) Filter by Name: Go
New Name Prefix... | Remove | Refresh

Name Prefix	Owner
myproject.com/pjct	dev_zhaox

图 9-39　查看已创建的命名前缀

9.3　NWDI 开发应用实例

以 NWDI 为基础的 Web Dynpro 开发包含下列步骤。

1）通过 NWDS 中的配置修改，导入开发配置。

2）创建开发组件，不仅限于 Web Dynpro 项目，也包括其他 NetWeaver 支持的项目类型。

3）完成开发和修改后，对开发内容进行检入和激活。

4）处理在代码同步过程中出现的错误。

9.3.1 导入开发配置

1）首先需要在开发工具 NWDS 中导入开发环境的配置。在导入之前，只有指定 SLD 的目录链接，才能根据 SLD 中对应的命名保留进行导入。打开 NWDS 后，选择“Window”→“Preferences”，如图 9-40 所示。

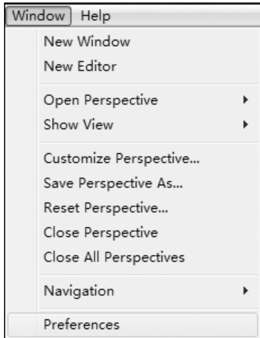


图 9-40 NWDS 配置的菜单选择

2）在弹出的偏好设置对话框中找到“Development Infrustrucure”→“System Landscape Directory”，也可在右上方输入关键字进行模糊搜索并定位，选择该配置后，在对话框右侧的 URL 栏中输入对应链接“http://local host:50100”，单击右侧的“Ping”按钮测试是否连接成功。确认链接填写无误后，单击“OK”按钮，如图 9-41 所示。

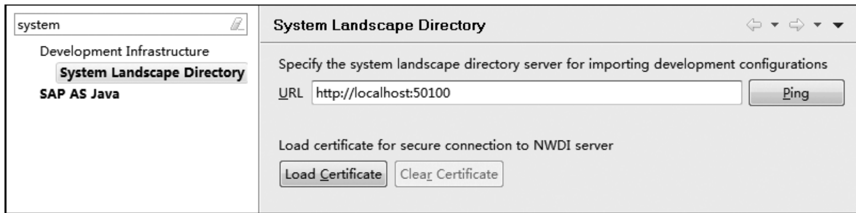


图 9-41 配置 SLD 的对话框

3）接着切换到“Development Infrastructure”，在空白处右击鼠标，选择“New”→“Development Configuration”，新建一个开发配置，如图 9-42 所示。

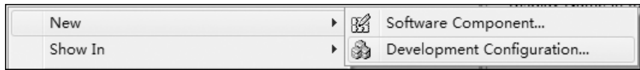


图 9-42 创建开发配置的菜单选择

4）在弹出的对话框中选择第一项“Import from System Landscape Directory(SLD)”，从已配置的 SLD 中导入开发轨道，随后单击“Next”按钮，如图 9-43 所示。

5）对话框中会列出所配置 SLD 中所有可用的轨道，选择“BPMD_dev”之后，所有的开发都会在该轨道中的软件组件中进行，如图 9-44 所示。该软件组件的包名已预定义好为“myproject. com”（即开发人员在该软件组件下建立的开发组件会存储在目录为“com. myproject”的文件夹中）。

6）随后按默认配置完成导入向导，对应的轨道就会出现在 NWDS 中的浏览选项卡中，如图 9-45 所示。



图 9-43　选择导入选项

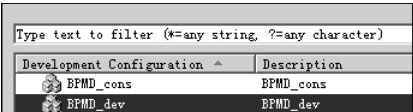


图 9-44　选择对应轨道



图 9-45　导入后效果

9.3.2　创建开发组件

1）展开“BPMD_dev”轨道，一般在开发工具与服务器版本一致的情况下，会看到与“LocalDevelopment”一致的目录结构。找到自定义创建的“BPMS”软件组件并右击，选择“New”→“Development Component”，创建一个新的 DC，在弹出对话框中选择要新建的 DC 的类型，随后单击“Next”按钮，如图 9-46 所示。

2）严格按照项目组命名规范，输入 DC 的名称以及标题等信息。建议选中“Keep DC Local for Now”复选框，暂时将开发组件保留在本地选项。由于新创建的开发组件修改比较频繁，因此在单个开发人员负责开发的情况下，没有必要每次新建 Java 文件或开发对象都进行创建活动并与 DTR 进行同步，等完成初步开发后，再一并手动进行提交，如图 9-47 所示。

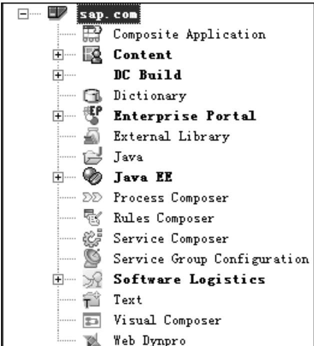


图 9-46　选择项目类型



图 9-47　输入项目信息

3）随后根据默认配置完成创建，即可在本地进行编码、编译、部署以及测试等工作。忽略具体的开发过程（可参考之前 Web Dynpro 开发介绍的章节），在完成第一个版本的开发后，需要将源代码同步至 DTR 进行保存，右击对应的 DC，并选择“Add to Source Control”，如图 9-48 所示。

4）在首次同步时，DTR 服务器监测到了开发对象修改，所以需要创建对应的活动。在弹出建立活动的对话框中输入必要的活动名称以及具体的描述，用于标识该活动的修改目的及相关简要信息，随后单击“OK”按钮，如图 9-49 所示。

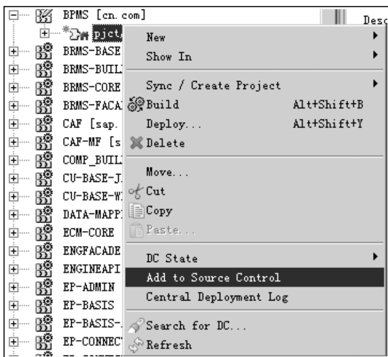


图 9-48　同步 DTR 的菜单选择

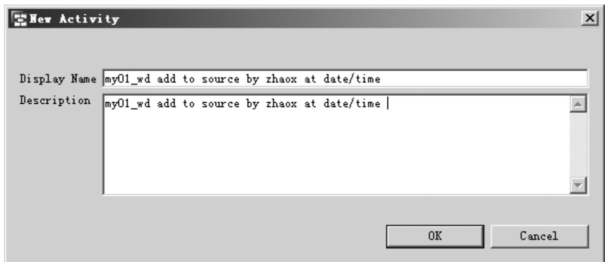


图 9-49　创建活动对话框

5）在创建活动后，需要继续步骤 3）中的添加动作。选择刚才创建的活动，单击“OK”按钮完成开发对象的添加，如图 9-50 所示。

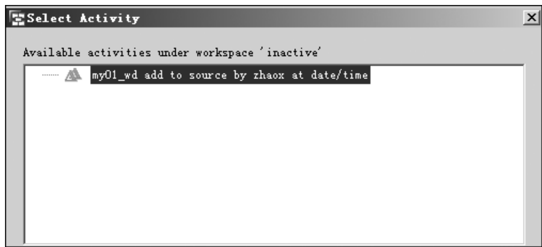


图 9-50　查看创建的活动

6）接着在对应的视图中展开活动，就能看到所有已修改的开发对象，包括源代码的文件夹、配置文件、源代码文件等内容，如图 9-51 所示。

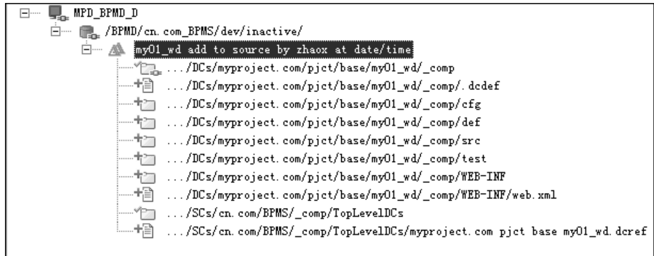


图 9-51　在活动中添加开发对象

9.3.3　检入与激活

1）在确认编译无误，可以上传至 DTR 进行编译后，首先进入“Open Activities”的视图，找到对应的活动，右击鼠标并选择“Checkin”进行检入，如图 9-52 所示。

2）在弹出的对话框中需要确认所选活动是否正确、包含的开发对象是否正确，确认无误后单击“OK”按钮，如图 9-53 所示。

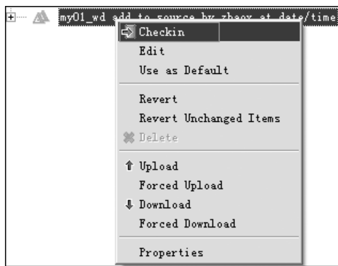


图 9-52　检入的菜单选择

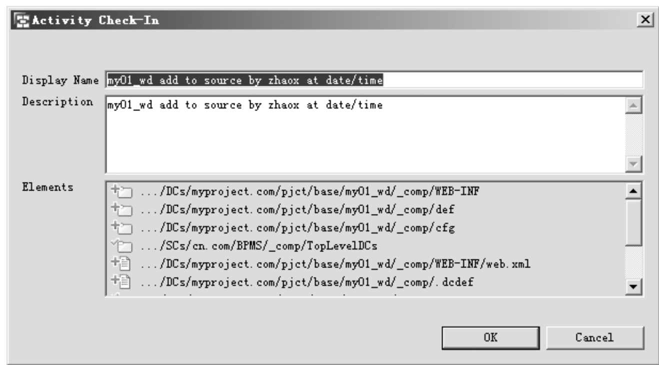


图 9-53　导入内容确认对话框

3）检入成功后，会有对话框提示是否自动将修改内容进行激活。如果有多个开发人员协同开发时，请谨慎选择自动激活选项。在本实例中，单击“Yes”按钮直接激活修改，如图 9-54 所示。



图 9-54　自动激活确认对话框

注：如果未弹出是否激活的提示对话框窗口，则可以通过偏好设置中的“Development Infrastructure”修改具体的开发工具配置。

4）在弹出的激活对应活动的确认对话框中，确认无误后，单击“Activate”按钮进行激活，如图 9-55 所示。

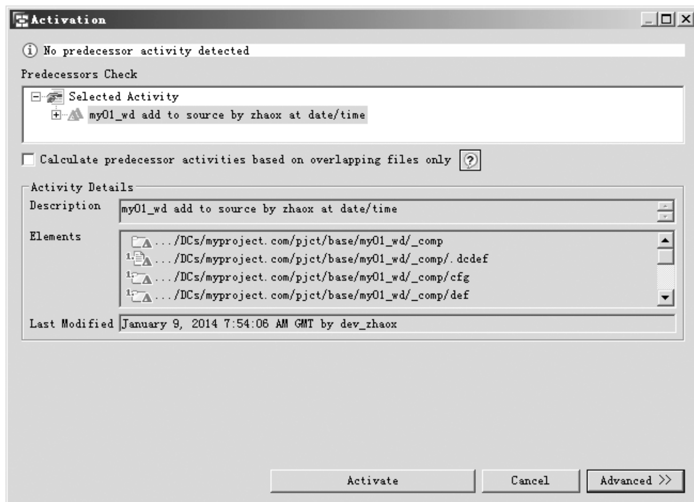


图 9-55　激活操作对话框

5）在 CBS 中会自动生成一个新的激活请求 ID，可以通过激活请求的视图查看具体的请求修改状态，单击“Open Request View”直接跳转至对应视图，如图 9-56 所示。

6）在视图中可以单击刷新按钮，刷新请求的最新状态，在对应的请求号右侧出现绿色对勾后，标识该激活请求已成功完成，如图 9-57 所示。



图 9-56　激活请求对话框

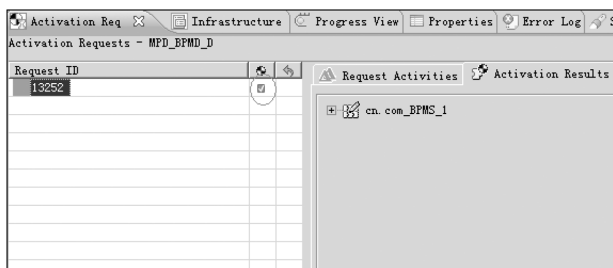


图 9-57　查看激活状态

如果激活失败，则单击“View Activation Log”按钮，查看激活操作日志或者查看 CBS 编译日志，并分析失败产生的原因。代码提交产生的问题应及时改正并重新激活，直至激活成功；其他依赖 DC 造成的激活失败，应联系 NWDI 管理员和相关开发人员解决。

7）激活成功后，才可以将对应的活动进行传输。首先进入传输视图中，找到已成功激活的活动，如图 9-58 所示。

8）右击对应的活动，并选择“Release”释放该活动，如图 9-59 所示。这与 ABAP 开发中激活成功后，需要进行释放请求的操作非常相似。

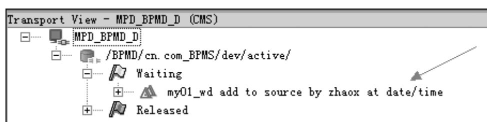


图 9-58　查看传输视图

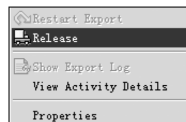


图 9-59　释放操作的菜单选择

9）在弹出的对话框中确认需要释放的活动的信息是否正确，随后单击“Release activities”按钮进行释放操作，如图 9-60 所示。

10）释放的活动将通过 CMS 的传输管理进行后续的系统间对象传输动作，如图 9-61 所示。

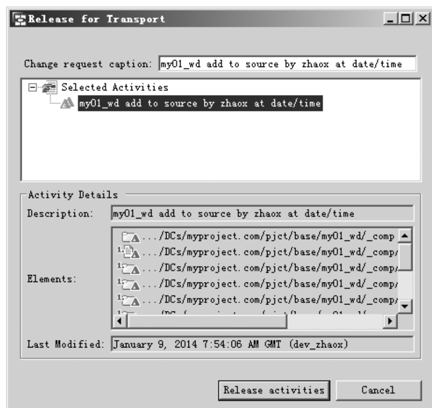


图 9-60　释放操作对话框

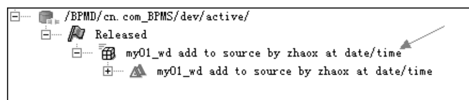


图 9-61　查看传输对象

9.3.4 协同开发场景

下面介绍在多个开发人员协同开发时的一些具体开发步骤。

开发实例：

1）准备两个工作空间以及两个具备“NWDI. DEVELOPER”权限的用户，首先使用其中一个用户登录 NWDI，并创建一个 Web Dynpro 类型的 DC 工程，如图 9-62 所示。

2）在 Web Dynpro 项目中创建对应的组件和视图，如图 9-63 所示。

3）在视图中添加一个按钮元素，并绑定“Query”自定义动作，如图 9-64 所示。

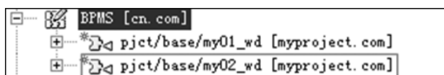


图 9-62　创建新开发组件

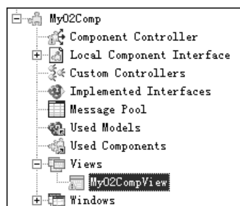


图 9-63　创建对应的组件和视图



图 9-64　创建按钮

4）根据之前的操作介绍，将该 Web Dynpro 项目的所有内容检入至 DTR 目录中，随后登录 DTR 并定位到该 Web Dynpro 项目目录中的视图控制器文件，单击最右侧的“下载”按钮并查看文件内容，如图 9-65 所示。

my02comp					
My02Comp.wdcomponent	dev_zhaox	application/xml	2179	11.01.2014 06:26:45 GMT	11.01.2014 06:32:51 GMT
My02Comp.wdcontroller	dev_zhaox	application/xml	718	11.01.2014 06:26:45 GMT	11.01.2014 06:32:51 GMT
My02CompInterface.wdcontroller	dev_zhaox	application/xml	760	11.01.2014 06:26:45 GMT	11.01.2014 06:32:51 GMT
My02CompInterface.wdlocalcompintfdef	dev_zhaox	application/xml	1374	11.01.2014 06:26:45 GMT	11.01.2014 06:32:51 GMT
My02CompInterfaceView.wdcontroller	dev_zhaox	application/xml	927	11.01.2014 06:26:45 GMT	11.01.2014 06:32:51 GMT
My02CompInterfaceView.wdinterfaceview	dev_zhaox	application/xml	934	11.01.2014 06:26:45 GMT	11.01.2014 06:32:51 GMT
My02CompMessagePool.wdmessagepool	dev_zhaox	application/xml	588	11.01.2014 06:26:45 GMT	11.01.2014 06:32:51 GMT
My02CompView.wdcontroller	dev_zhaox	application/xml	1202	12.01.2014 07:09:43 GMT	12.01.2014 07:10:15 GMT
My02CompView.wdview	dev_zhaox	application/xml	2619	12.01.2014 07:09:43 GMT	12.01.2014 07:10:15 GMT

图 9-65　查看 DTR 明细

5）文件内容中含有新建的名称为“Query”的按钮动作，如图 9-66 所示。

```
- <Controller.Actions>
  - <Action name="Query">
    - <IncomingEvent.EventHandler>
        <Core.Reference path="EventHandler:onActionQuery" />
      </IncomingEvent.EventHandler>
    </Action>
  </Controller.Actions>
```

图 9-66　查看按钮动作定义

6）在 DTR 上该文件的上次修改时间，即活动的检入时间可以通过激活明细进行查看。另外，还可以查看不同开发阶段检入的历史版本。

7）接着回到 NWDS 开发工具中，切换至“Development Infrastructure”视图，在文件浏览中找到“src”目录中的视图控制器开发对象文件，如图 9-67 所示。

8）右击该文件，并选择“Team”→“Show In”→“Version Graph”，显示历史版本，如图 9-68 所示。

9）接着就可以查看对应文件的的 DTR 检入版本，如图 9-69 所示。注意，这里的不同

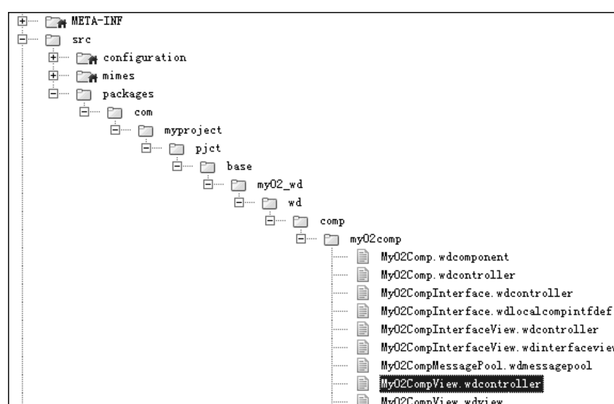


图 9-67　浏览开发对象目录

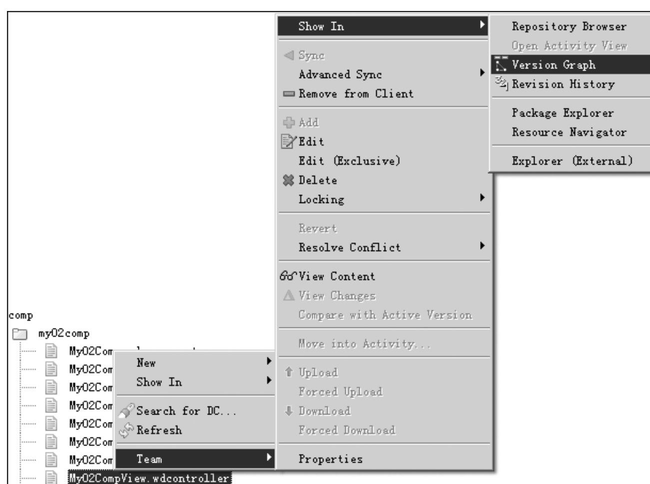


图 9-68　显示历史版本的菜单选择

版本并不是 CBS 的激活历史，而是“Checkin”检入的历史版本。

10）图 9-69 中每一个结点代表一次检入操作，右击每一个结点可以查看具体检入活动的明细内容，包含该版本下的文件内容、该版本与上一版本的内容差异、该版本与 DTR 最新版本的内容差异和同步为对应版本等内容，如图 9-70 所示。

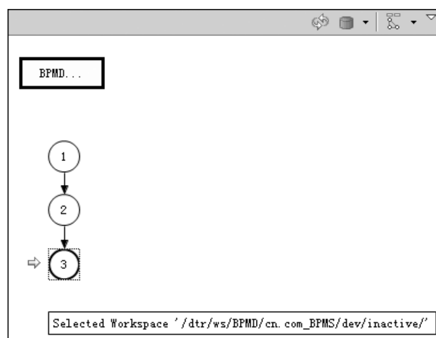


图 9-69　DTR 中的版本记录

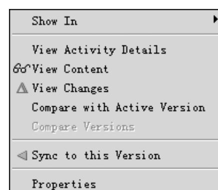


图 9-70　检入相关的菜单选择

11）在左侧的文件列表中，右击视图控制器的文件，并选择“Team”→“Show In”→“Revision History”。该视图是对“Version Graph”版本图形的列表显示，显示了未激活工作空间中的激活历史，行项目的具体内容与版本图形中的结点一致，如图 9-71 所示。

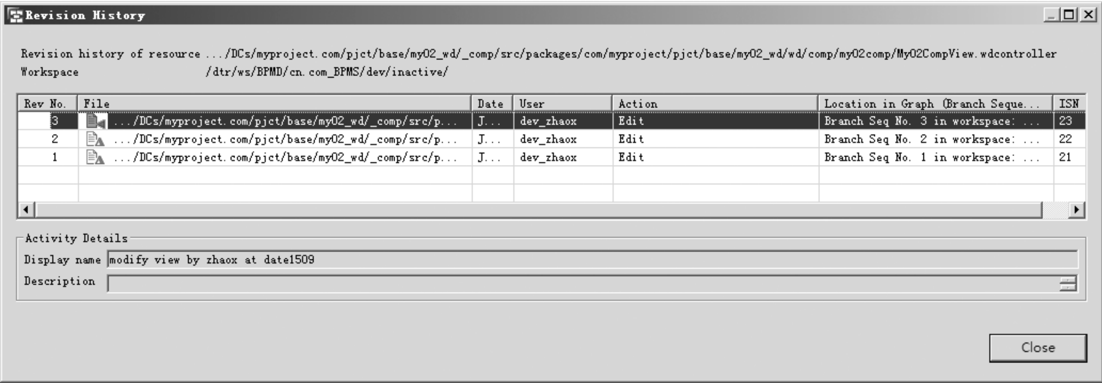


图 9-71　显示恢复历史

12）接着切换到另外一个 NWDS 的开发空间，并且使用不同的账号登录 NWDI，用于模拟另一个开发人员需要进行协同开发的场景。在打开 NWDS 并登录同一个 NWDI 后，找到之前自定义配置的“BPMS”软件组件，以及在之前步骤中创建的 Web Dynpro 项目，右击该项目并选择“Sync/Create Project”→“Create Project”，将 DTR 中的项目复制创建至本地的工作空间中，如图 9-72 所示。



图 9-72　同步项目的菜单选择

13）接着进入对应的 Web Dynpro 视图，找到之前创建的视图，如图 9-73 所示。

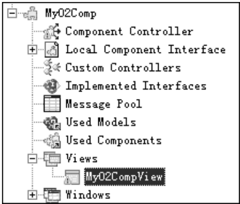


图 9-73　同步后的目录结构

14）当再次编辑该视图时，在按钮的事件处理器中添加下列代码：

```
publicvoid onActionQuery( com. sap. tc. webdynpro. progmodel. api. IWDCustomEvent wdEvent)
        {
//@ @ begin onActionQuery( ServerEvent)
        wdComponentAPI. getMessageManager( ). reportSuccess( "执行成功!" ) ;
//@ @ end
        }
```

15）完成修改后，单击“保存”按钮，将自动弹出“Checkout”对话框，因为 DTR 已监测到当前的开发正在进行一个新的开发修改，单击“Check Out”按钮即可开始修改，如图 9-74 所示。

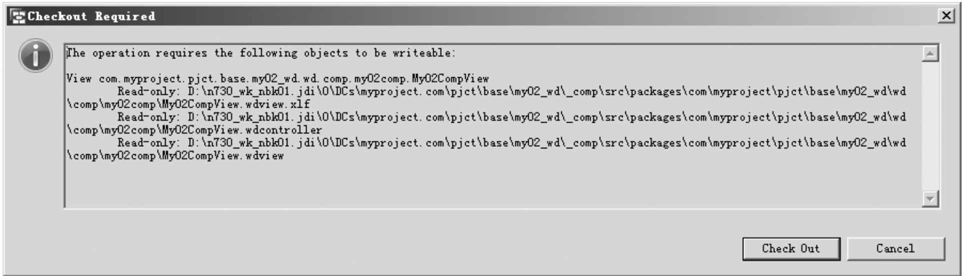


图 9-74　检出确认对话框

16）接着选择需要包含修改内容的动作，如没有修改中的活动，则可以单击“New”创建一个新的活动来包含将要进行的开发修改内容，如图 9-75 所示。

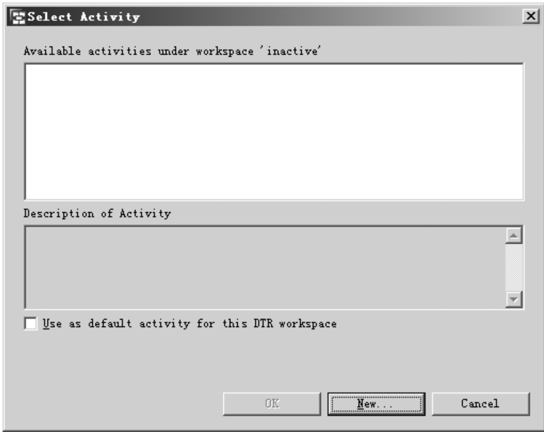


图 9-75　新建活动

17）在弹出的创建活动对话框中输入必要的描述（描述要素一般包含修改人、修改时间和修改内容等），便于后期的监控管理和维护，如图 9-76 所示。

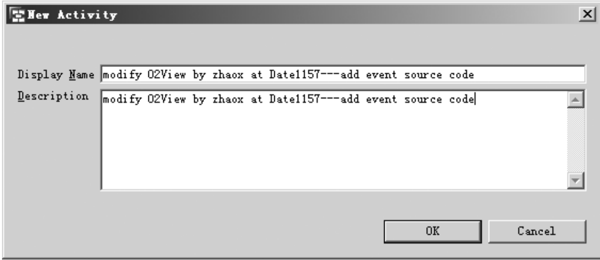


图 9-76　输入活动信息

18）完成活动创建后，右击活动进行检入，并对 DTR 中开发对象进行同步更新，如图 9-77 所示。

19）检入后，选择自动激活，同时生成一个新的激活请求，在激活请求视图可以查看具体的激活状态，如图 9–78 所示。



图 9–77　活动检入的菜单选择

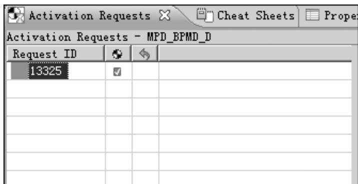


图 9–78　查看激活请求视图

20）请求包含的活动中新增了上一次未激活的活动，每次激活前都会按时间顺序检测 CBS 之前是否有未激活的活动，可以方便进行一并提交，避免出现与 DTR 服务器端不同步的现象，如图 9–79 所示。

21）切换到传输管理视图，两个工作空间下已激活活动目录中均出现了待释放的活动，但分别对应的是两位开发人员各自修改的内容。使用第一个开发人员的账号登录的工作空间查看的是第一次创建 Web Dynpro 项目时的活动，如图 9–80 所示。

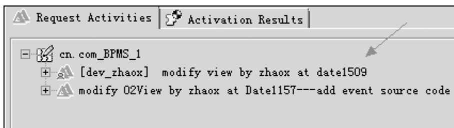


图 9–79　查看请求包含的内容



图 9–80　第一个开发人员查看的活动

22）而使用第二个开发人员的账号登录后，只能看到他本人修改产生的活动内容。

23）活动的释放与 CBS 激活的原则保持一致，最新的活动释放时会优先释放之前的活动，如图 9–82 所示。

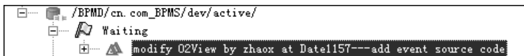


图 9–81　第二个开发人员查看的活动

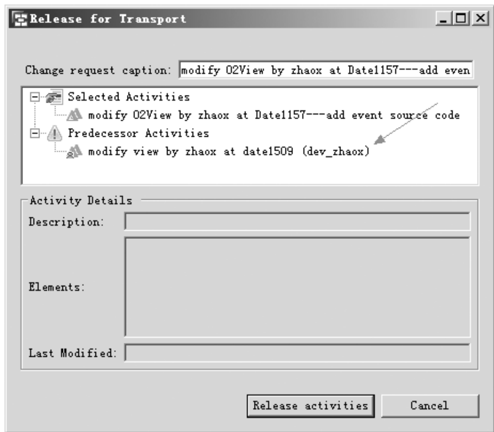


图 9–82　查看释放涉及的活动内容

24）在释放后，使用第一个开发人员的账号登录工作空间，并且编辑同一个视图，将会有错误消息提醒，提示文件不能被检出。这是因为本地的代码没有与服务端同步，之前使用了第二个开发人员的账号对同一个视图进行了修改并进行了检入，如图 9–83 所示。

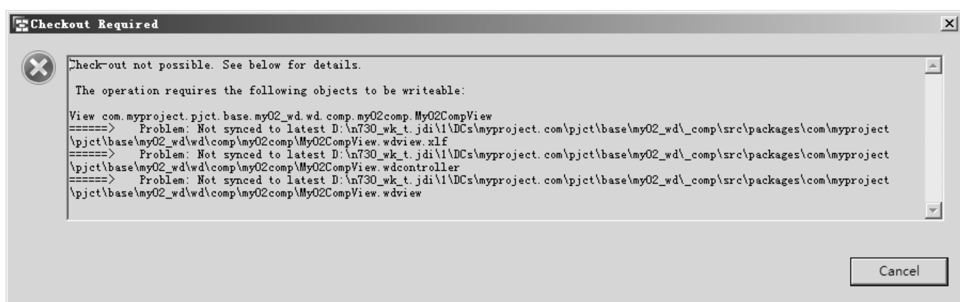
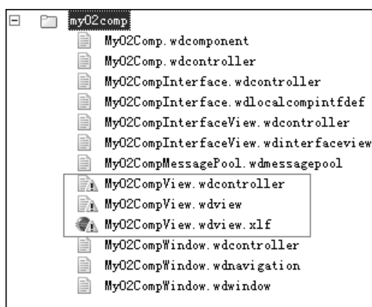


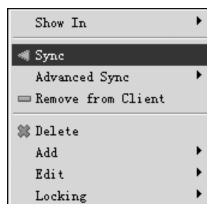
图 9-83　错误消息提醒

25）作为第一位开发人员，首先需要做的是同步源代码文件，将本地的开发对象与DTR 中最新版本进行同步，如图 9-84 所示。

26）右键对应的目录，选择“team”→“sync”进行同步操作，如图 9-85 所示。

27）同步完毕后，在文件目录中的“警告”标识的感叹号消失了，如图 9-86 所示。





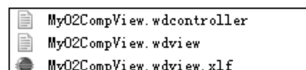


图 9-84　提醒不同步的开发对象　　图 9-85　同步操作的菜单选择　　图 9-86　同步后的开发对象

28）此时查看 Web Dynpro 项目的视图控制器，代码未更新，如图 9-87 所示。

```
public void onActionQuery(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionQuery(ServerEvent)
  //@@end
}
```

图 9-87　查看同步后的代码

29）这是由于同步至本地后，还未更新至 NWDS 开发工具中。右击 Web Dynpro 项目，并选择“Reload”，从本地文件目录重新读取开发对象，如图 9-88 所示。

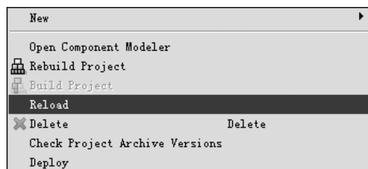


图 9-88　重装项目的菜单选择

30）重新载入后，即可看到最新修改后的源代码，如图 9-89 所示。

注：采用任何方式进行 DTR 代码同步后，需要在本地开发工具中对项目进行“Reload”操作，防止因未加载更新代码而带来修改的冲突。

```
public void onActionQuery(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionQuery(ServerEvent)
    wdComponentAPI.getMessageManager().reportSuccess("执行成功!");
  //@@end
}
```

图 9-89　查看重装后的代码

9.3.5　代码冲突

在实际的项目开发中，多个开发人员协同开发确实能够提升项目的开发效率，但在不同开发人员对同一个 DC 进行修改时，往往会因为对相同版本的开发对象检出并修改而产生代码冲突。下面通过实例解析处理版本冲突的方式。

开发实例：

1）继续以上一节中创建的 Web Dynpro 项目为例，再次编辑视图时，首先进行检出动作，如图 9-90 所示。

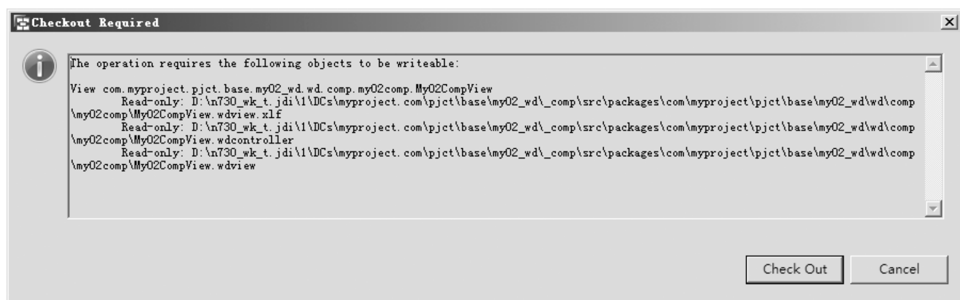


图 9-90　检出对话框

2）检出后，将新修改的内容放入一个新的活动中，如图 9-91 所示。

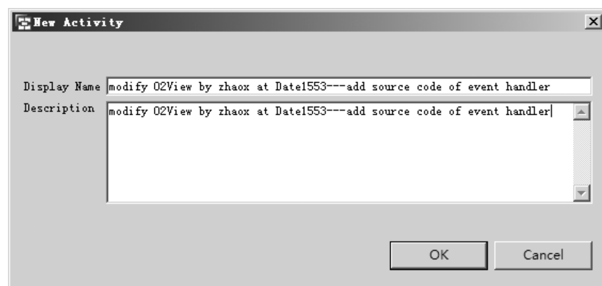


图 9-91　创建活动

3）接着修改按钮的实现方法，添加下列代码，作为第一名开发人员的修改记录：

```
publicvoid onActionQuery(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent)
    {
//@@begin onActionQuery(ServerEvent)
    wdComponentAPI.getMessageManager().reportSuccess("执行成功!");
    wdComponentAPI.getMessageManager().reportSuccess("本账户的修改!");
//@@end
}
```

4）使用第二名开发人员的账号登录 NWDI，并使用新的开发工具的工作空间进行修改，在检出时 NWDI 首先会弹出确认框提示该文件已经被其他用户检出并在修改。在实际开发过程中，需要特别关注这个提醒消息，并及时与组内其他开发人员进行沟通，以便顺利进行后续工作。在本实例中，为了测试代码冲突，仍然单击“OK”按钮，继续检出，如图 9-92 所示。

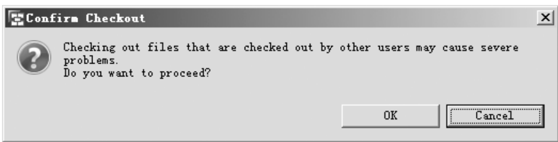


图 9-92　代码冲突确认对话框

5）同样地，创建一个动作，用于包含第二名开发人员对开发对象所做的修改，如图 9-93 所示。

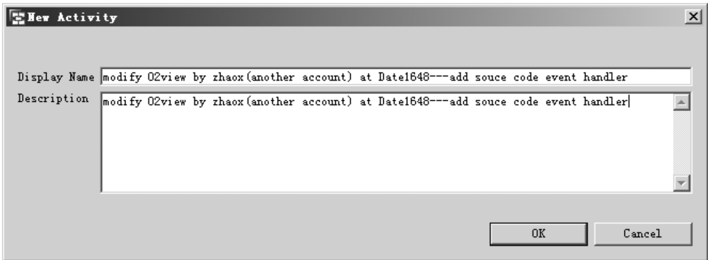


图 9-93　创建动作

6）接着修改按钮的实现方法，添加下列代码，作为第二名开发人员的修改记录：

```
publicvoid onActionQuery(com. sap. tc. webdynpro. progmodel. api. IWDCustomEvent wdEvent)
    {
//@ @ begin onActionQuery(ServerEvent)
    wdComponentAPI. getMessageManager(). reportSuccess("执行成功!");
    wdComponentAPI. getMessageManager(). reportSuccess("其他账号的修改!");
//@ @ end
    }
```

7）此时，两名开发人员都已对同一个开发对象进行了修改，任何人进行“Checkin”的检入操作都会失败，如图 9-94 所示。

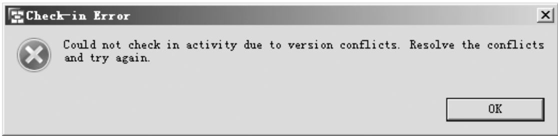


图 9-94　报错对话框

8）在 DTR 视图中可以看到文件已标记为红色，表示已有代码冲突产生，如图 9-59 所示。

9）如果右击这些源代码文件，并查看版本图像，则可以看到最新修改中产生了两个同样的版本，如图 9-96 所示。

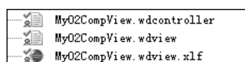


图 9-95　产生冲突的开发对象

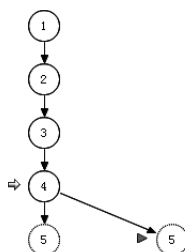


图 9-96　查看 DTR 版本记录

10）正因为两个版本的存在，所以导致第二名开发人员在检入活动时失败。比较简单的两种处理方式：一种方法是强制接受本地修改，覆盖服务器端代码；另一种方法是接受服务端的版本，覆盖本地代码，并做再次修改提交。如果比较熟悉处理代码冲突，还可以使用“Merge”功能，手动将两个版本代码进行合并。这里使用第二种方法，首先重新从服务器端同步源代码文件，右击活动中产生冲突的开发对象，选择“Resolve Conflict”→“Accept Active（Remote）Version”，如图 9-97 所示。

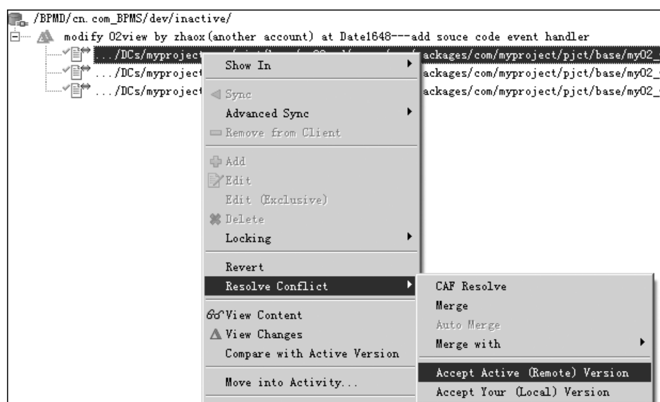


图 9-97　处理冲突的菜单选择

11）弹出的对话框确认是否接受覆盖本地代码的操作，单击“Accept Active Version”按钮，如图 9-98 所示。



图 9-98　确认版本的对话框

12）在将所有文件覆盖后，记得将项目进行“Reload”重新载入最新版本的代码，接着进入视图控制器，就可以看到第一名开发人员已经添加的代码，如图 9-99 所示。

```
public void onActionQuery(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionQuery(ServerEvent)
    wdComponentAPI.getMessageManager().reportSuccess("执行成功!");
    wdComponentAPI.getMessageManager().reportSuccess("本账户的修改!");
  //@@end
}
```

图 9-99　查看重装后的代码

注：在团队开发过程中，不论使用哪种方法处理代码冲突，一定要提前与其他开发人员或开发组长进行沟通，团队协作不仅依赖于系统提供的检入、检出、覆盖等功能，还依赖于团队成员的良好沟通机制。

9.4 开发组件的依赖

在之前 Web Dynpro 的章节中，介绍了 Web Dynpro 的对外可见部分仅有它的接口视图和接口控制器，通过它们可以将不同的 Web Dynpro 项目串联在一起；而在 SAP NetWeaver 平台的开发中，每一个开发组件的对外可见部分是它的公共部分，对于不同类型的项目依赖和关联，都是通过公共部分来进行串联。

例如，在第 7 章中，在将 BPM 项目与 Web Dynpro 项目进行关联之前，首先需要在 Web Dynpro 中定义公共部分，并将需要公布的组件添加至其中，配置完成后，在 BPM 的业务流程和任务配置中才能找到对应的 Web Dynpro 的接口视图和流转触发事件。

9.4.1 公共部分与依赖明细

在 NWDI 中，所有开发组件都可以定义两种类型的公共部分，如图 9-100 所示。

- Compilation，设计时使用，主要为其他 DC 在开发时提供 API，其他 DC 使用此 DC 的类及方法时，需要依赖于本 DC 的公共部分。
- Assembly：运行时使用，本身不能独立部署运行，需要打包到其他 DC 时建立此种类型公共部分，其中应包含接口及其实现，并保证其功能运行正常。

而在 DI 透视图中，选中任意一个公共部分后，还需要配置具体的开发组件依赖明细（Dependency Details），它包含了以下 4 种方式（见图 9-101）：

- Design Time：用于此 DC 在开发设计时，某些插件对于其他 DC 组件基于此种类型的依赖关系进行引用。
- Build Time：用于此 DC 在编译时，需要其他 DC 的类及组件时，建立此种类型依赖关系。
- Deploy Time：用于此 DC 引用其他 DC 的类及组件，并且需要约束在此 DC 部署时，依赖的 DC 必须已经部署，建立此种类型依赖关系。
- Runtime：用于此 DC 在运行时需要引用其他 DC 的类及组件时，建立此种类型依赖关系。

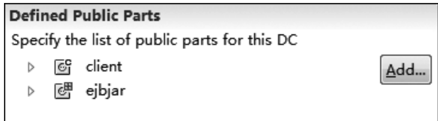


图 9-100 公共部分的两种类型

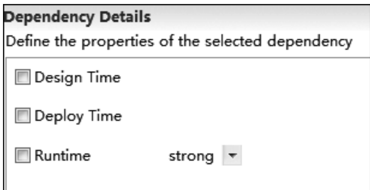


图 9-101 依赖明细

注：如果所依赖的 DC 没有部署至服务器，当前 DC 可以部署但无法正确运行，只有所有依赖的 DC 成功部署后，当前 DC 才可以正常运行。

9.4.2 依赖规则参考

开发中经常涉及 DC 之间的依赖使用，DC 之间的依赖需要遵守固定的规则，以防止因依赖问题而导致的运行时错误或异常。

除了 WD4J 的项目以外，任何包含“Compilation”与“Assembly”类型的公共部分的开发组件（DC）在被其他开发组件依赖时都仅有一次对编译时（Build Time）的选中，“Compilation”类型公共部分确保编译时能够正常，而“Assembly”类型公共部分是运行时被使用，任何一个错误的依赖都会导致设计时或运行时的失败，而通过对编译时的重复限制会避免因多处编译而导致重复的类加载异常。下面将以表的方式展现不同项目类型中的公共部分的定义名称和包含实体，在表中通过“X”表示勾选对应的依赖关系，留空则为不勾选依赖关系。

1. EJB 项目

EJB 公共部分定义参考（见图 9-102）见表 9-3。

表 9-3　EJB 公共部分定义参考

类　型	名　称	包含实体
Compilation	client	项目的接口（interface）和工厂类（factory class）
Assembly	ejbjar	相关 JAR 文件，即 ejb - jar

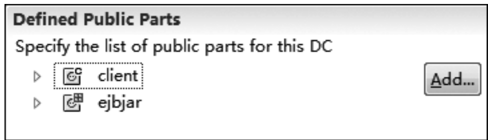


图 9-102　公共部分定义

EAR 项目对于 EJB 的依赖配置参考（见图 9-103）见表 9-4。

表 9-4　EAR 项目对于 EJB 的依赖参考

	Build Time	Design Time	Deploy Time	Runtime
EJB 项目				
ejbjar	X			
client				



图 9-103　依赖配置

WD4J 项目对于 EJB 的依赖配置参考（见图 9-104）见表 9-5。

表 9-5　WD4J 项目对于 EJB 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EJB 项目				
ejbjar				
client	X			

WD4J 项目对于 EAR 的依赖配置参考（见图 9-105）见表 9-6。

表 9-6　WD4J 项目对于 EAR 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EAR 项目				X

2. LIB 项目

LIB 公共部分定义参考（见图 9-106）见表 9-7。

表 9-7　LTB 公共部分定义参考

类　型	名　称	包 含 实 体
Compilation	api	项目包含的 JAR 文件
Assembly	archives	项目包含的 JAR 文件

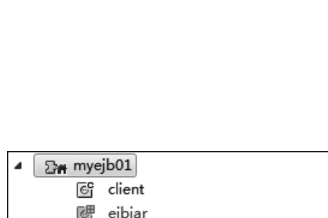


图 9-104　依赖配置

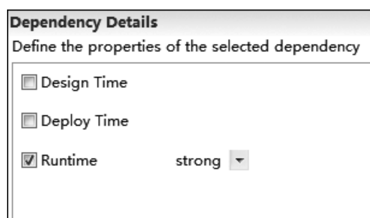


图 9-105　依赖明细



图 9-106　公共部分定义

EAR 项目对于 LIB 的依赖配置参考（见图 9-107）见表 9-8。

表 9-8　EAR 项目对于 LIB 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EJB 项目				
archives	X			
api				

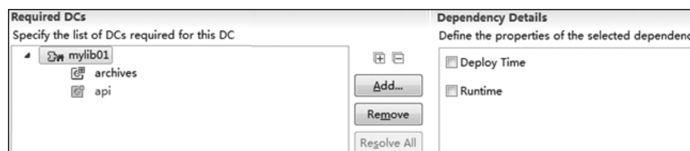


图 9-107　依赖明细

WD4J 项目对于 LIB 的依赖配置参考（见图 9-108）见表 9-9。

表 9-9　WD4J 项目对于 LIB 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EJB 项目				
archives				
api	X			

WD4J 项目对于 EAR 的依赖配置参考（见图 9-109）见表 9-10。

表 9-10　WD4J 项目对于 EAR 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EAR 项目				X

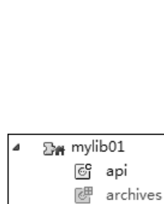


图 9-108　依赖配置

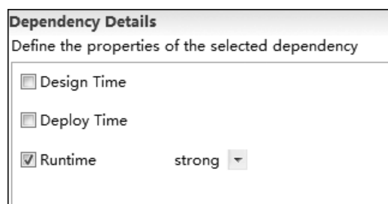


图 9-109　依赖明细

3. Java 项目

Java 项目公共部分定义参考（见图 9-110）见表 9-11。

表 9-11　Java 项目公共部分定义参考

类　型	名　称	包含实体
Compilation	api	项目的接口（interface）和工厂类（factory class）
Assembly	archives	项目的接口（interface）和工厂类（factory class）

EAR 项目对于 Java 的依赖配置参考（见图 9-111）见表 9-12。

表 9-12　EAR 项目对于 Java 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EJB 项目				
archives	X			
api				

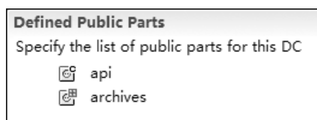


图 9-110　公共部分定义

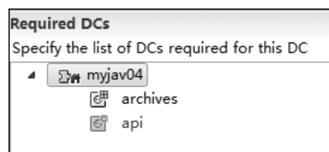


图 9-111　依赖配置

WD4J 项目对于 Java 的依赖配置参考（见图 9-112）见表 9-13。

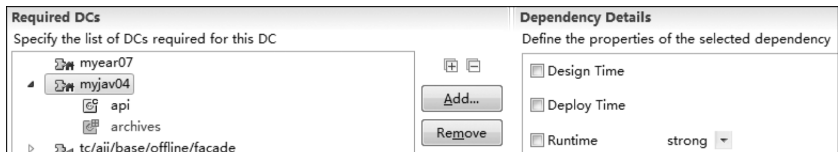


图 9–112　依赖明细

表 9–13　WD4J 项目对于 Java 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EJB 项目				
archives				
api	X			

WD4J 项目对于 EAR 的依赖配置参考（见图 9–113）见表 9–14。

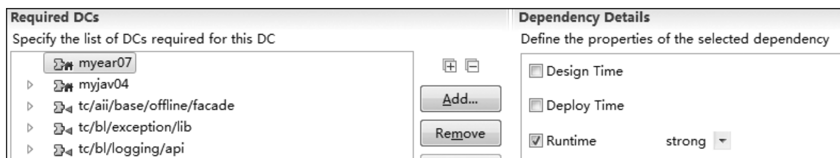


图 9–113　依赖明细

表 9–14　WD4J 项目对于 EAR 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EAR 项目				X

4. EAR 项目之间

EAR 项目对于 EAR 的依赖配置参考（见图 9–114）见表 9–15。



图 9–114　依赖明细

表 9–15　EAR 项目对于 EAR 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
EAR 项目				X

5. WD4J 项目

WD4J 项目公共部分定义参考（见图 9–115）见表 9–16。

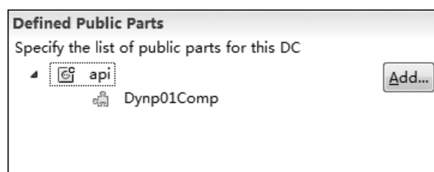


图 9–115　公共部分定义

表 9-16　WD4J 项目公共部分定义参考

类　　型	名　　称	包含实体
Compilation	api	项目中的组件

BPM 项目对于 WD4J 的依赖配置参考（见图 9-116）见表 9-17。



图 9-116　依赖明细

表 9-17　BPM 项目对于 WD4J 的依赖配置参考

	Build Time	Design Time	Deploy Time	Runtime
WD4J 项目				X
api	X			

WD4J 项目对于 WD4J 的依赖配置参考（见图 9-117）见表 9-18。

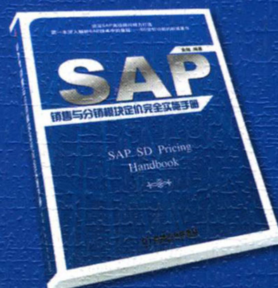


图 9-117　依赖明细

表 9-18　WD4J 项目对于 WD4J 的依赖配置参考

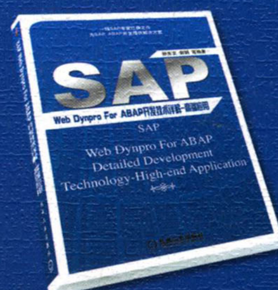
	Build Time	Design Time	Deploy Time	Runtime
WD4J 项目				X
api	X			

进阶阅读



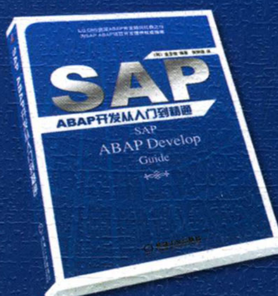
《SAP 销售与分销模块定价完全实施手册》

ISBN.978-7-111-50993-6



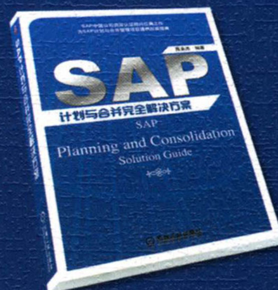
《SAP Web Dynpro For ABAP开发技术详解-高端应用》

ISBN.978-7-111-54749-5



《SAP ABAP开发从入门到精通》

ISBN 978-7-111-41700-2



《SAP计划与合并完全解决方案》

ISBN 978-7-111-39913-1

ISBN 978-7-111-58177-2

目标读者群：

- 本书可作为Web Dynpro开发人员的速查手册，也可作为正在使用相关技术的最终用户以及其他技术顾问学习和了解Web Dynpro技术的参考书。

本书详细介绍了Web Dynpro for Java的相关开发技术、底层架构及周边拓展应用，以实例贯穿全书，配以丰富的理论说明。前半部分集中对Web Dynpro for Java开发技术、底层架构进行详细的描述和讲解，不仅说明了怎么开发，更阐述了为什么这样开发。后半部分主要介绍与Web Dynpro for Java相关的各类前沿技术实例以及在新版本中的应用，包括新版本的门户集成、NWA管理者平台配置、BPM流程管理建模开发、NWDI集成开发环境等内容。

SAP

上架指导：ERP/SAP

51CTO学院
为中华崛起传播智慧

地址：北京市百万庄大街22号
邮政编码：100037

电话服务
服务咨询热线：010-88361066
读者购书热线：010-68326294
010-88379203
网络服务
机工官网：www.cmpbook.com
机工官博：weibo.com/cmp1952
金书网：www.golden-book.com
教育服务网：www.cmpedu.com
封面无防伪标均为盗版


机械工业出版社
微信公众号


计算机分社
微信订阅号



策划编辑◎丁诚 / 封面设计◎道乐文化

定价：99.00元